CS1010S Programming Methodology

Lecture 11

Visualising Data

4 Nov 2020

What this lecture is not

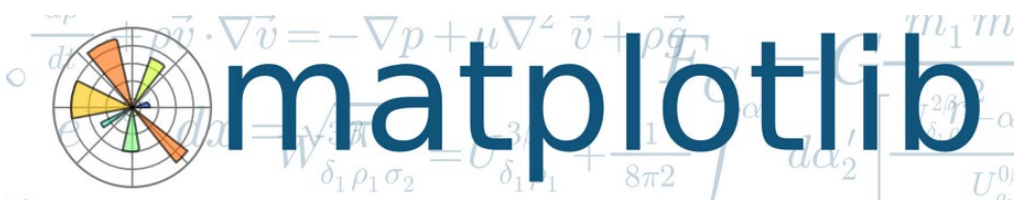
- What charts should be used
- How to analyse data

What this lecture is

- Sanitising raw data
- Formatting data
- Creating the chart

- Unlocks your creativity
  - Not be limited by lack of tools

Common visualisation libraries



matplotlib

```
from matplotlib import pyplot as plt

x = tuple(range(100))
y = tuple(map(lambda x:x*x, x))

fig, ax = plt.subplots()

ax.plot(x, y)
fig.show()
```

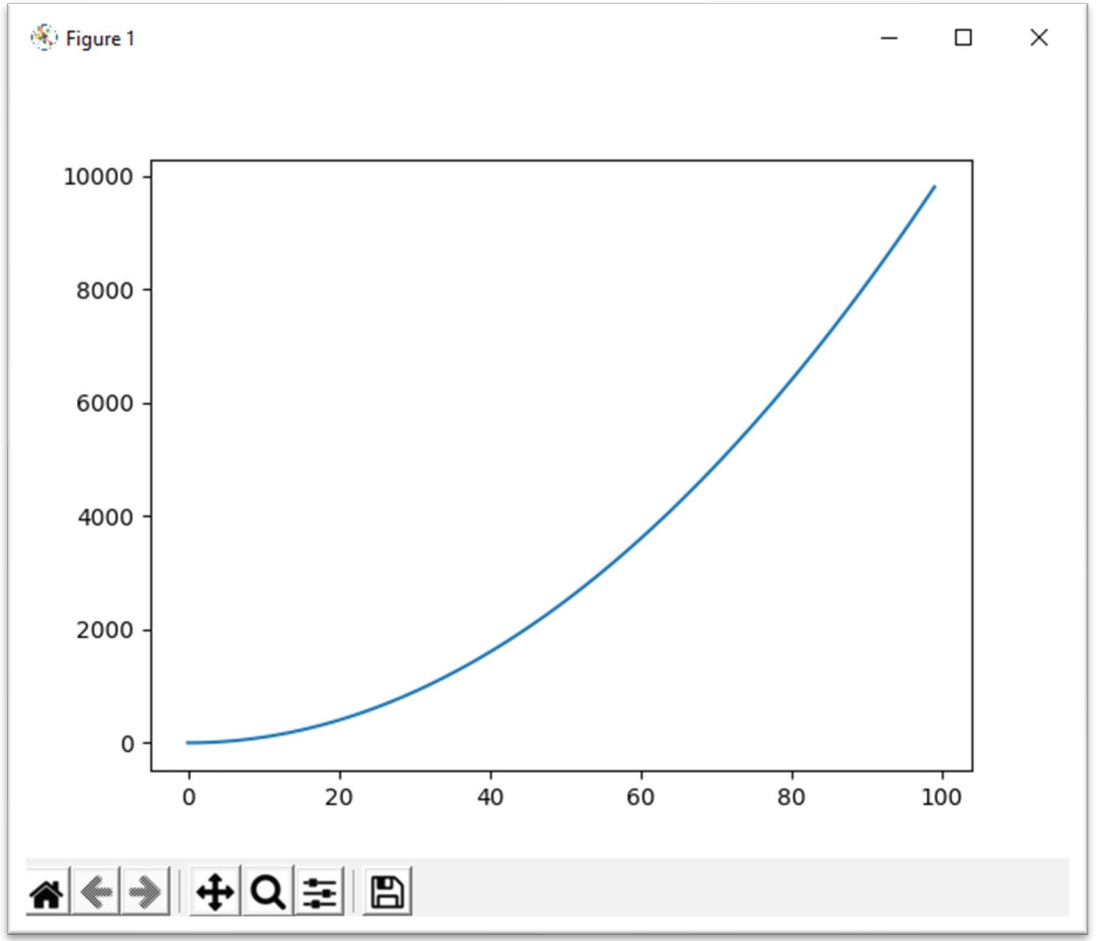


Figure and Axis

```
fig, ax = plt.subplots()
```

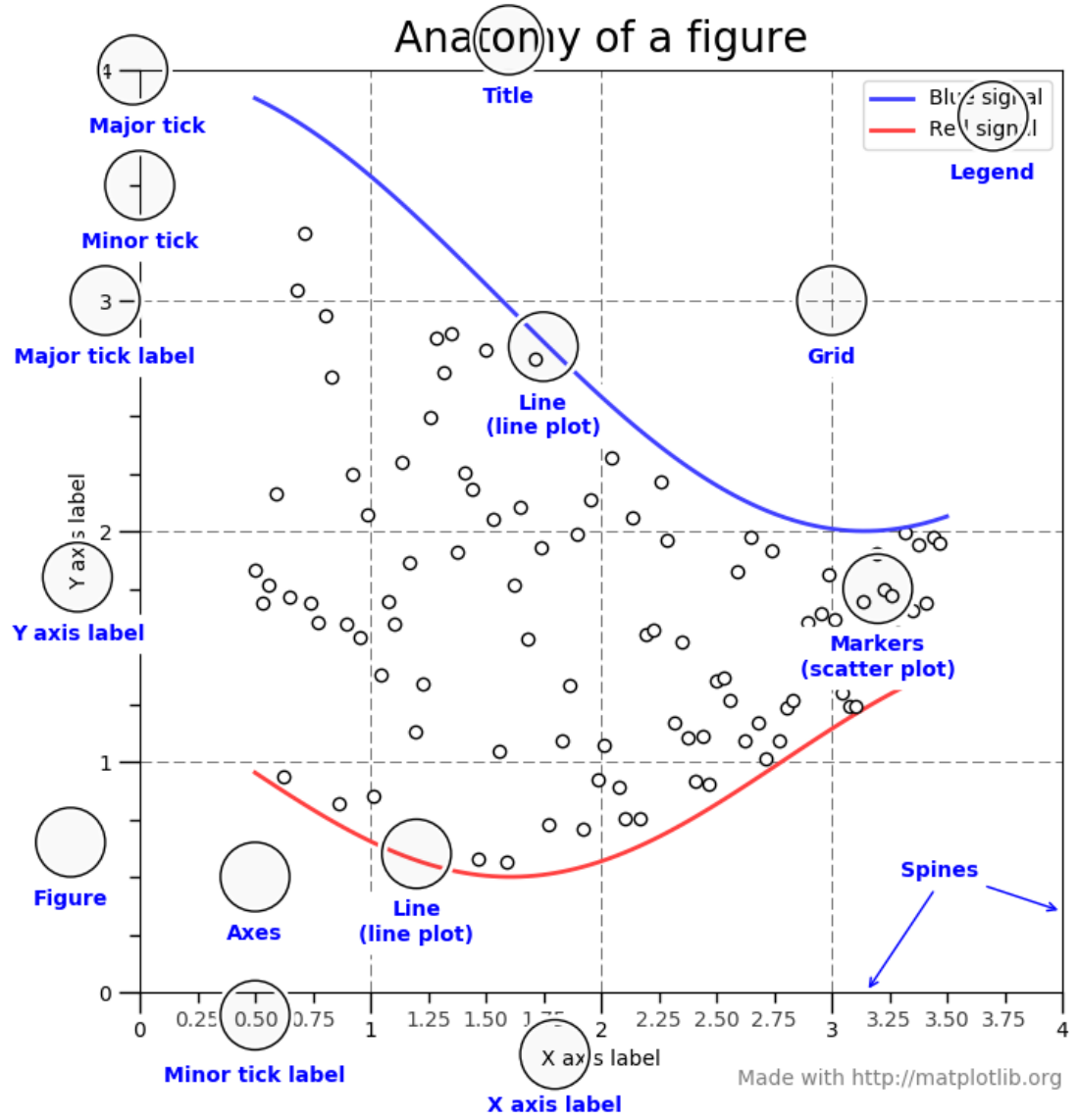
The whole figure
i.e. the window

The "plot area"
in the figure

A figure can contain multiple plots

```
fig, axes = plt.subplots(2, 2)
```

A 2x2 grid of axes



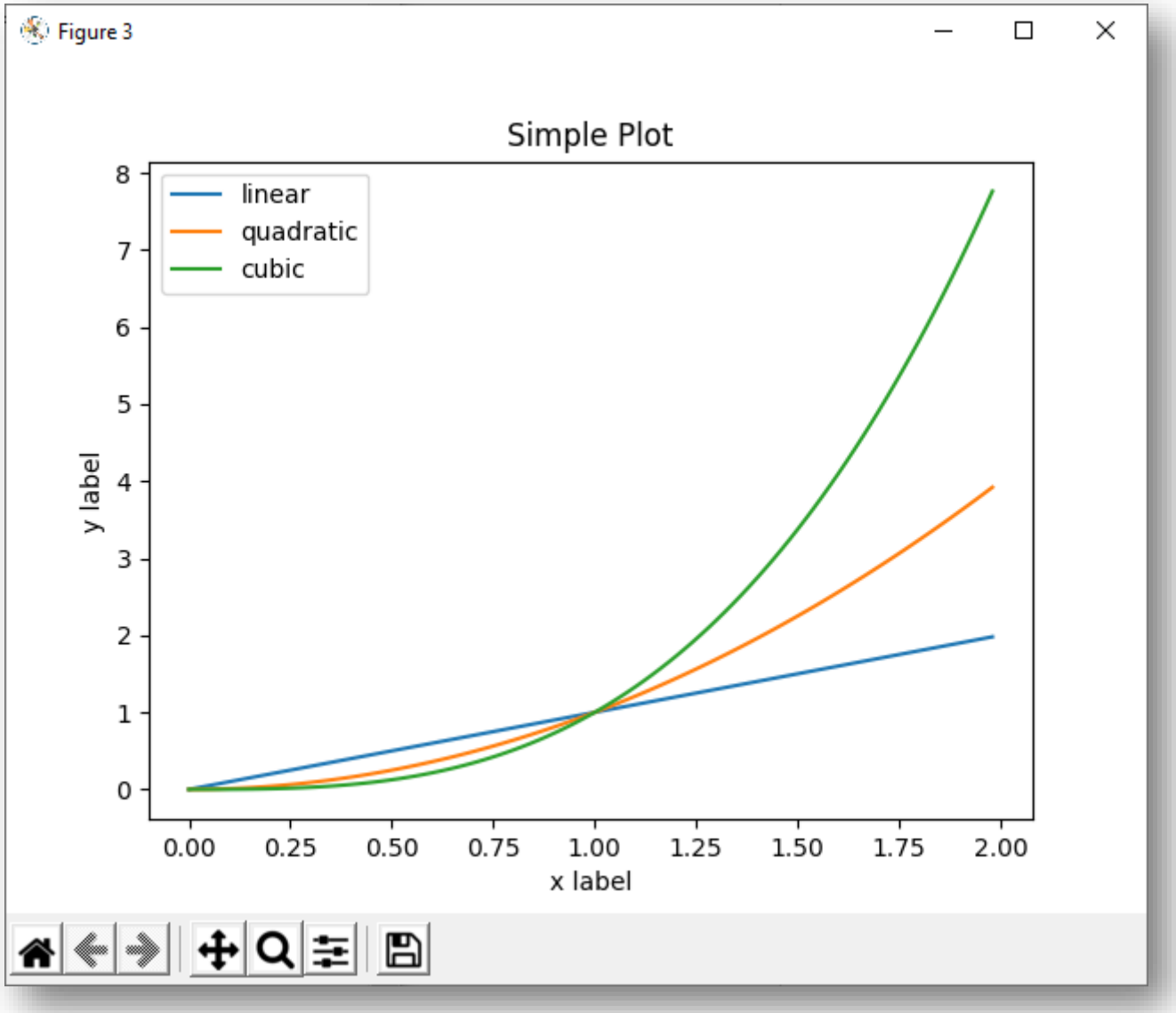
Figure

Simple Plot

```
x = [x/50 for x in range(100)]

fig, ax = plt.subplots()
ax.plot(x, x, label='linear')
ax.plot(x, [x**2 for x in x], label='quadratic')
ax.plot(x, [x**3 for x in x], label='cubic’)

ax.set_xlabel('x label’)      # Add an x-label to the axes.
ax.set_ylabel('y label’)      # Add a y-label to the axes.
ax.set_title("Simple Plot")   # Add a title to the axes.
ax.legend()                   # Add a legend.
```

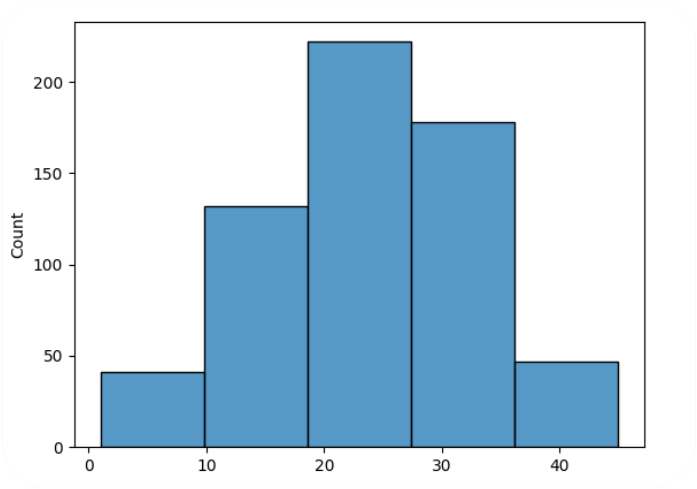
Simple Plot

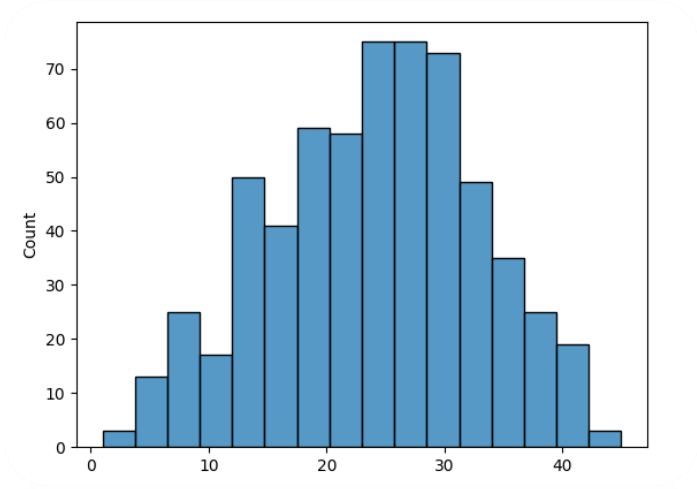
```
fig.show()
```

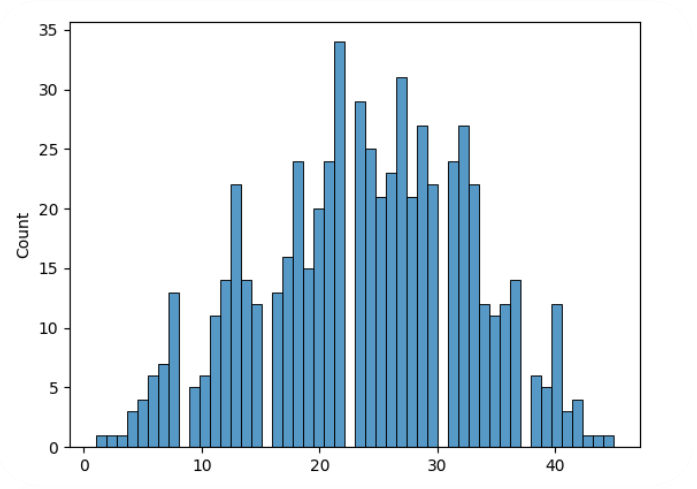


Midterm Marks Distribution

- Let's start with something simple (and practical)
  - Visualize distribution of marks
  - Using a Continuous Distribution Function
  - Because histograms are too *basic*

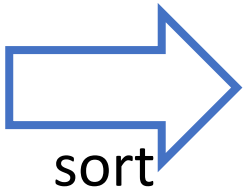




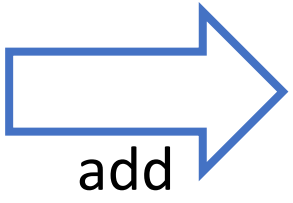


Quick ’n Dirty CDF

Marks
20
15
17
35
40
4
⋮
36
7



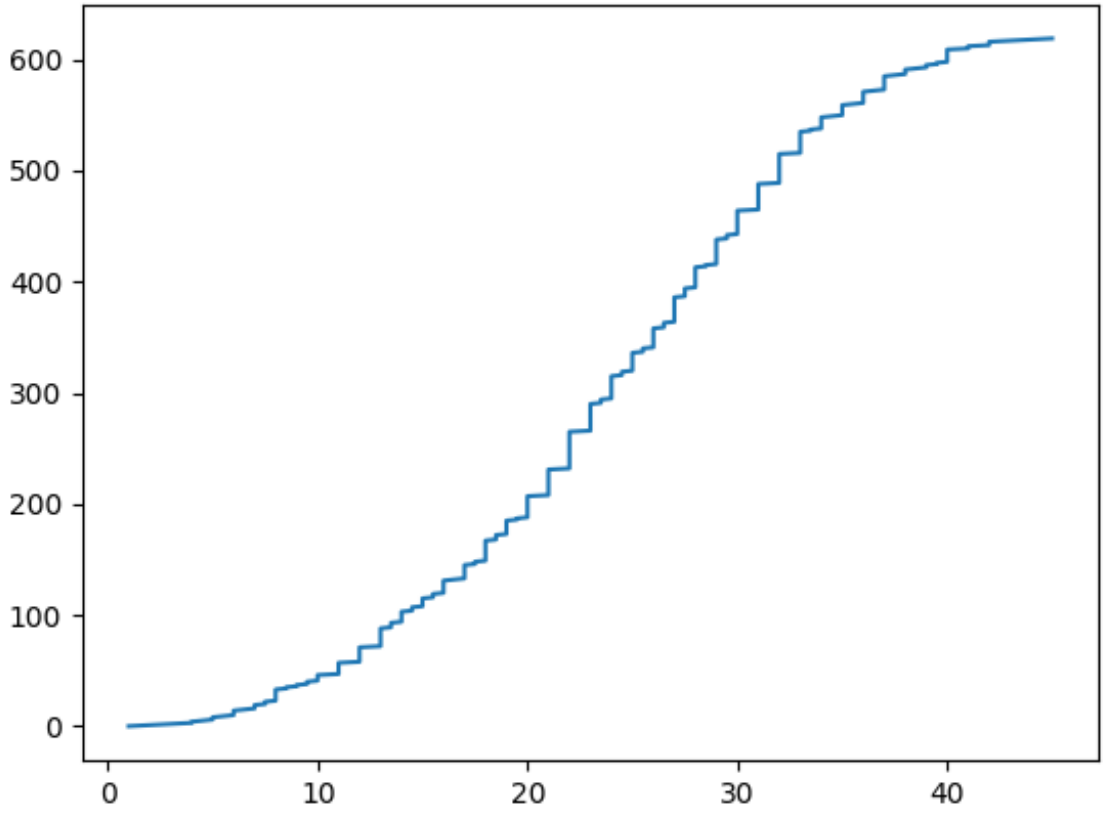
Marks
2
3
4
4
5
5
⋮
48
48



Index	Marks
0	2
1	3
2	4
3	4
4	5
5	5
⋮	⋮
619	48
620	48

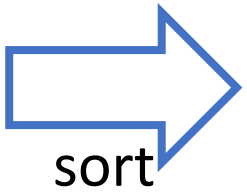
Quick ’n Dirty CDF

```
marks = [20, 15, 17, 35, 60 . . . # list of marks

marks.sort()

fig, ax = plt.subplots()

ax.plot(marks, range(len(marks)))
fig.show()
```

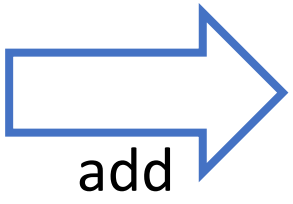


Quick ’n Dirty CDF

Marks
20
15
17
35
40
4
⋮
36
7



Marks
2
3
4
4
5
5
⋮
48
48

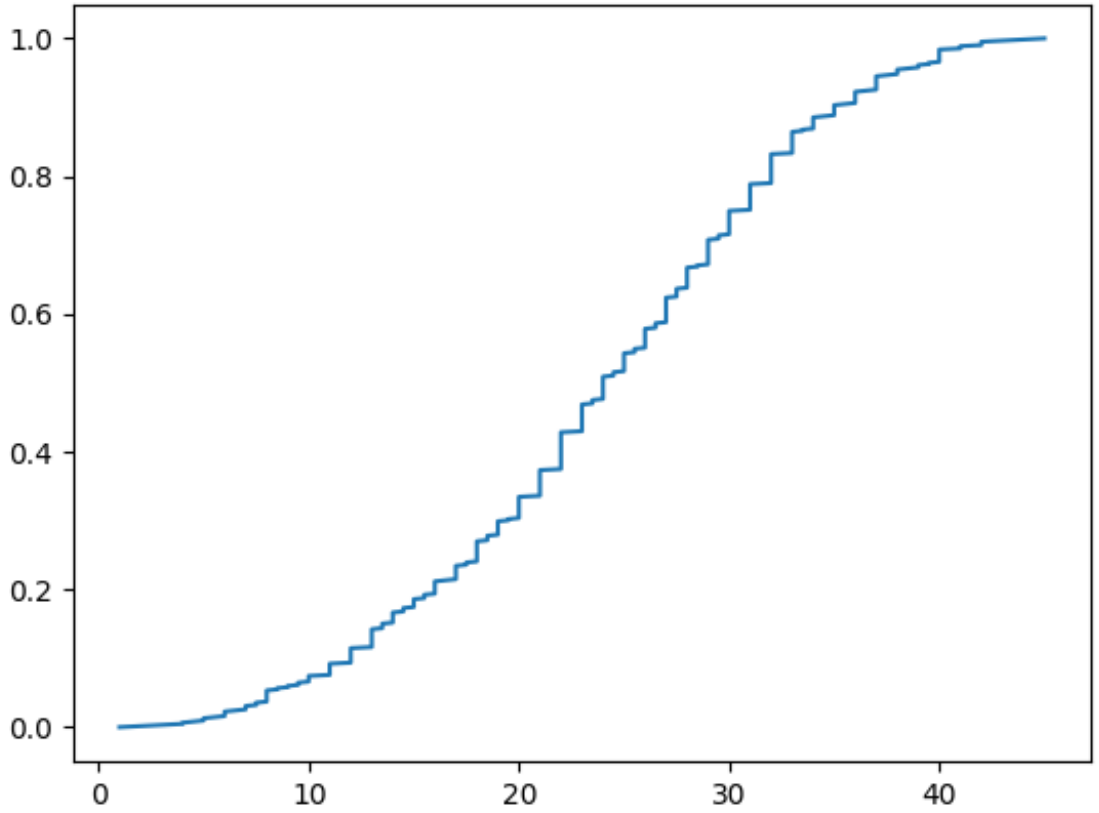


Index	Marks
0	2
1	3
2	4
3	4
4	5
5	5
⋮	⋮
619	48
620	48

Index	Marks
0	2
0.0016	3
0.0032	4
0.0048	4
0.0064	5
0.0081	5
⋮	⋮
0.9984	48
1.0	48

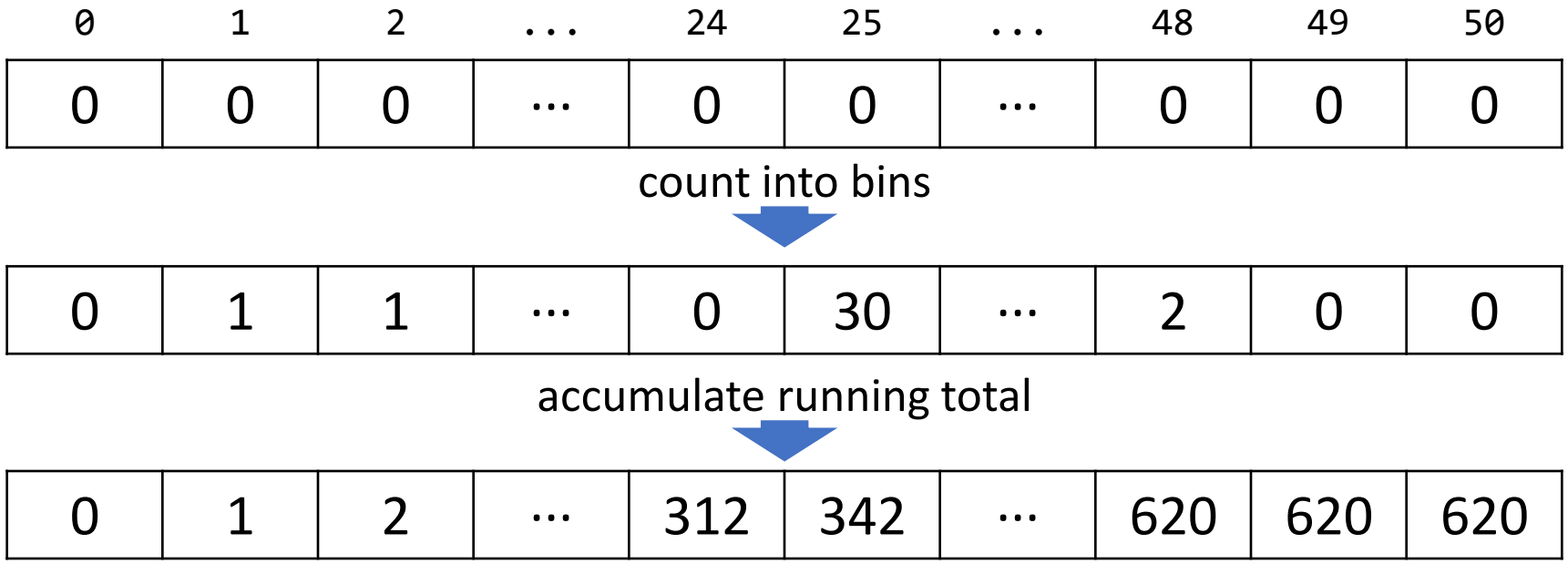
Quick ’n Dirty CDF

```
marks = [20, 15, 17, 35, 60 . . . # list of marks

marks.sort()

fig, ax = plt.subplots()

ax.plot(marks,
        [x/(len(marks)-1) for x \
         in range(len(marks))])
fig.show()
```



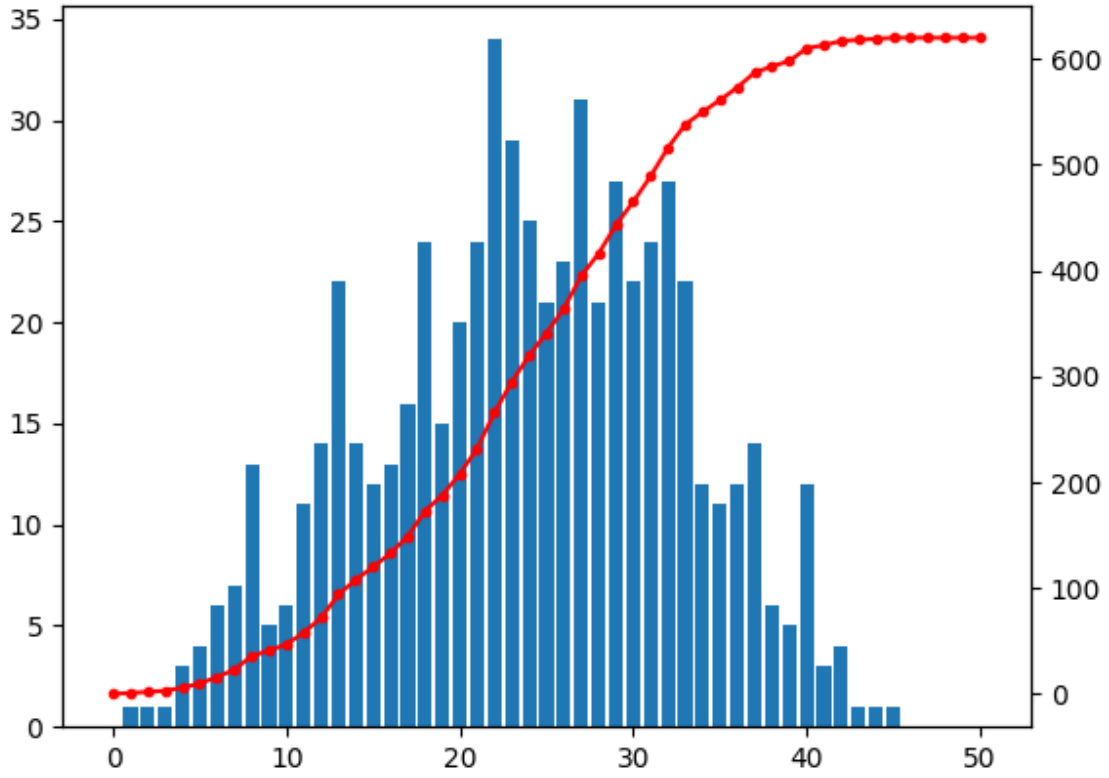
Problem with Quick ‘n Dirty

- There are n number of points plotted
  - Can be slow if there are a lot (millions) of points
- But there are only 50 marks
  - We only need 50 points
- Idea:



Better CDF

```
bins = [0] * 51    # 0 to 50 marks
for mark in marks:
    bins[mark] += 1

# now bins is a histogram
ax.bar(range(51), bins)

# running total
for i in range(1, 51):
     bins[i] += bins[i-1]
ax.twinx().plot(range(51), bins, 'r.-')
```



Plotting is easy when
data is well formatted

Issue: dealing with raw data

Wide-form vs Long-form

- Wide-form
  - Columns and rows contains different levels of data
- Long-form
  - Each variable is a column
  - Each records is a row

	month											
year		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
	1949	112	118	132	129	121	135	148	148	136	119	104	118
	1950	115	126	141	135	125	149	170	170	158	133	114	140
	1951	145	150	178	163	172	passengers	199	184	162	146	166	
	1952	171	180	193	181	183	218	230	242	209	191	172	194
	1953	196	196	236	235	229	243	264	272	237	211	180	201

year	month	passengers
1949	Jan	112
1949	Feb	118
1949	Mar	132
1949	Apr	129
1949	May	121
⋮	⋮	⋮
1953	Nov	180
1953	Dec	201

Wide-form vs Long-form

- Which form do we need?
  - For a year, passengers against month

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166

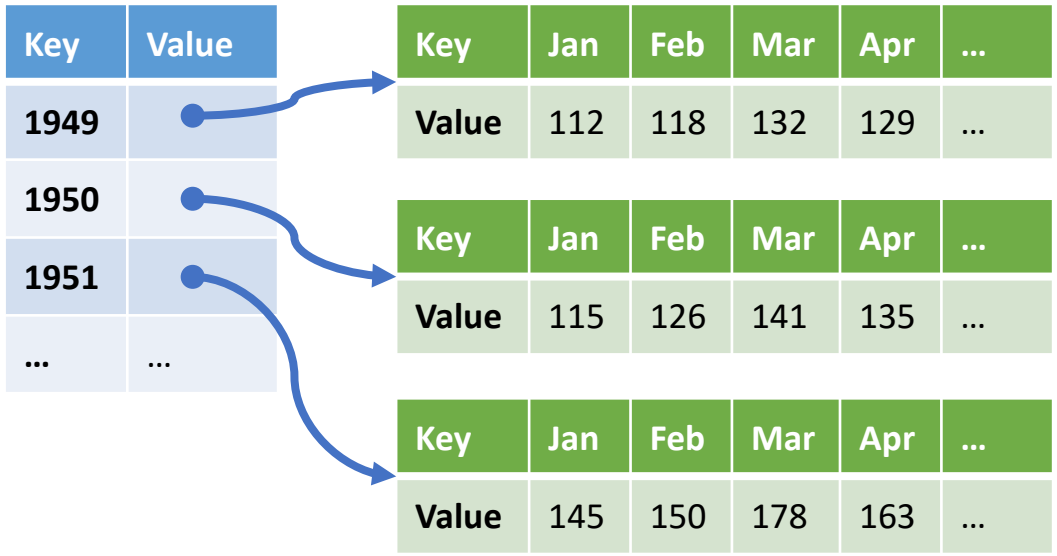
  - For a month, passenger against year

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166

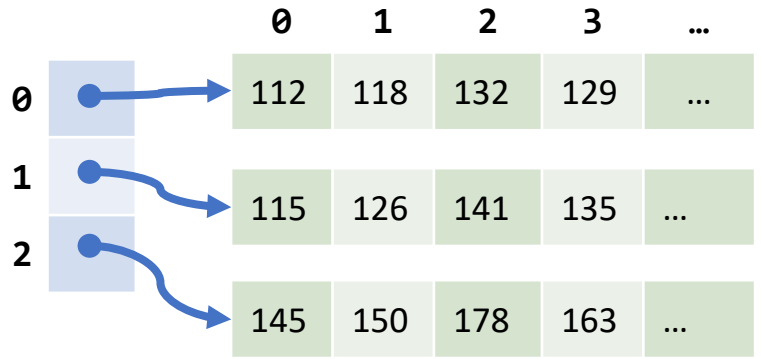
Pivoting

- To convert from long form to wide form
  - Usually the two parameters we are interested in

1. Decide on our wide-form structure
   - A table, using dictionary or list



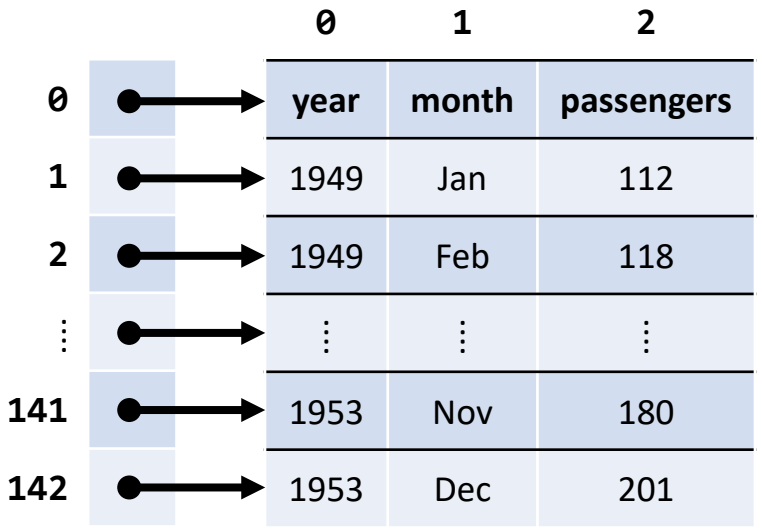
Dictionary-of-dictionaries



List-of-lists

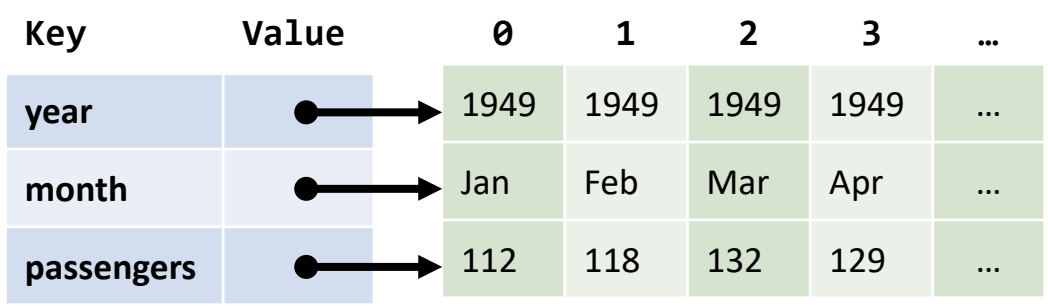
Pivoting

2. Determine the structure of the source data
   - A dictionary-of-dictionary, or list-of-lists, or combi



	0	1	2
0	year	month	passengers
1	1949	Jan	112
2	1949	Feb	118
⋮	⋮	⋮	⋮
141	1953	Nov	180
142	1953	Dec	201

List-of-lists



Key	Value	0	1	2	3	…
year		1949	1949	1949	1949	…
month		Jan	Feb	Mar	Apr	…
passengers		112	118	132	129	…

Dictionary-of-lists

Pivoting Example

- Long-form is a list-of-lists
  - Usually when read using csv reader

```
[[1949, 'Jan', 112], [1949, 'Feb', 118],
 [1949, 'Mar', 132], [1949, 'Apr', 129], …
]
```

- Wide-form is a dict-of-dicts
  - Usually with categorical data (not sequential)

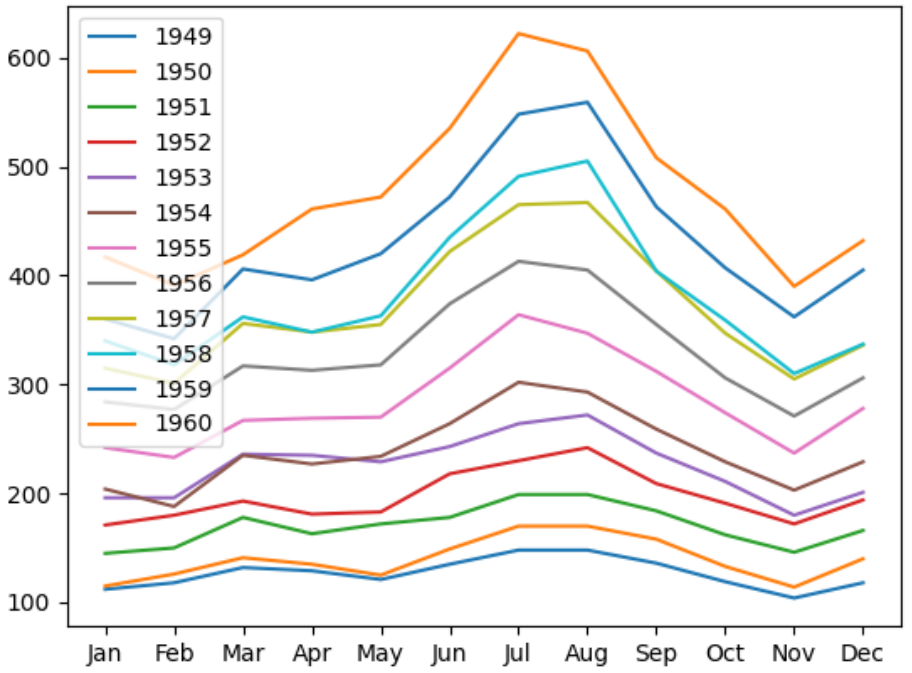
```
{1949: {'Jan': 112, 'Feb': 118, 'Mar': 132, 'Apr': 129, … }
 1950: {'Jan': 115, 'Feb': 126, 'Mar': 141, 'Apr': 135, … }
}
```

Pivoting Example

```
rows = read_csv(filename) # assume input is list-of-list
del rows[0]  # remove header

table = {}
months = ['Jan', 'Feb', 'Mar', 'Apr', ...

for year, month, pax in rows:
    if year not in table:
        table[year] = dict(map(lambda x:(x, 0), months))
    table[year][month] += pax
```

Pivoting Example

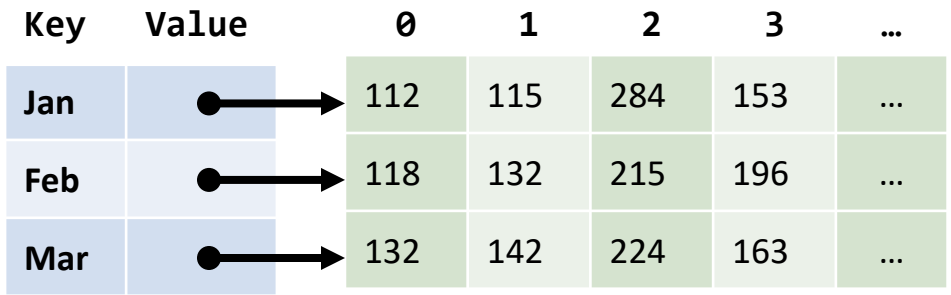
```
fig, ax = plt.subplots()

for year, row = table.items():
    ax.plot(row.keys(), row.values(), label=year)
ax.legend()

fig.show()
```



Box-plot

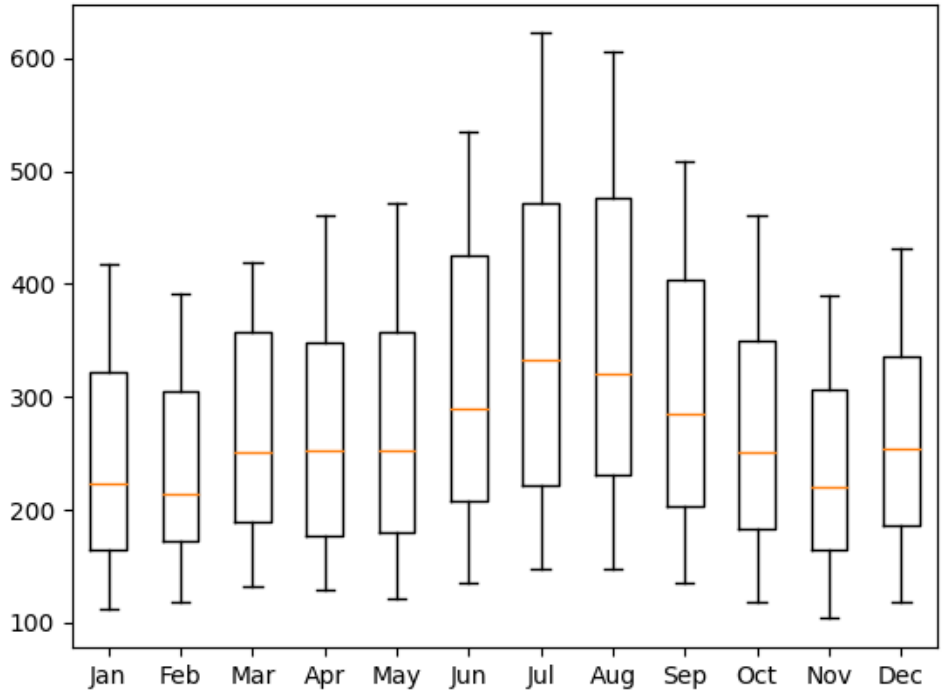
- For each month
  - mean (average), standard deviation
  - median, 25th and 75th percentile
  - maximum, mininum

- Idea: Just collect all values for each month in a list



Pivoting for Box-plot

```
table = {}

for year, month, pax in rows:
    if month not in table:
        table[month] = []
    table[month].append(pax)

# find average
avg = sum(table[1949])/len(table[1949])

ax.boxplot(table.values(), labels=table.keys())
```



“Messy”-form

- Long-form and Wide-form are “tidy”
  - Variables clearly defined in rows and/or columns
- Messy-form
  - Column are values, not variables
  - Multiple variables in one column
  - Variables in both rows and columns
  - Multiple types of observational units in the same table
  - A single observational unit in multiple tables

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software, 59*(10), 1 - 23. doi:http://dx.doi.org/10.18637/jss.v059.i10

Simple Steps

1. Determine what structure of data you need
2. Examine the structure of data given
3. Write code to pivot or melt

Tip:

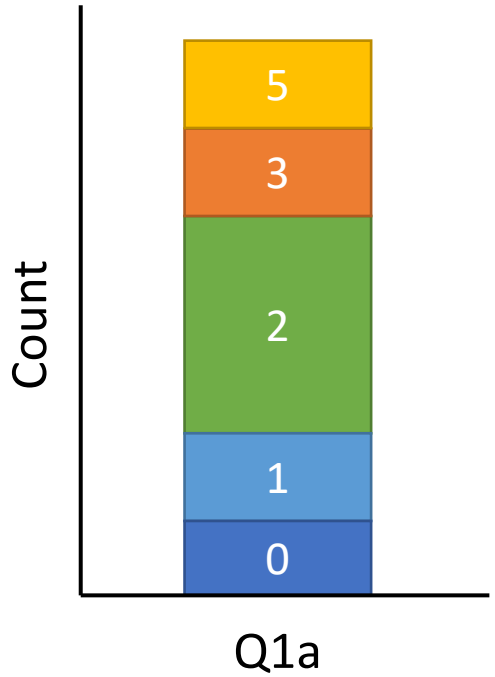
- Converting messy-form to long-form before pivoting may help

Back to Midterm Test

- What is the marks distribution for each question?
  - Stacked box plot

- What does matplotlib need?
  - Sequence for each category (mark)
  - Elements represent x-axis category

	Q1a	Q1b	Q1c	Q2a	…
0 marks	30	120	32	153	…
1 mark	118	132	64	43	…
2 marks	132	142	32	163	…



Source data

- What we get

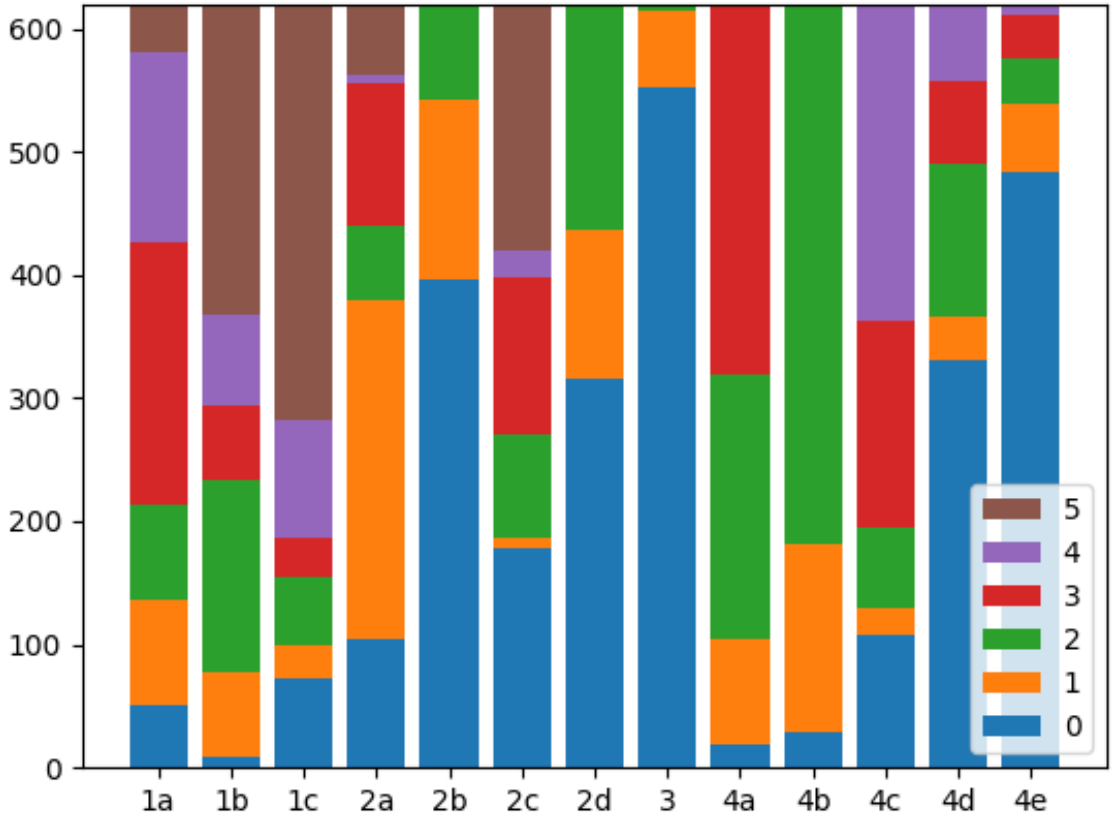
ID	Q1a	Q1b	Q1c	Q2a	Q2b	Q2c	Q2d	Q3	Q4a	Q4b	Q4c	Q4d	Q4e
E000001	5	1	5	5	2	5	2	3	3	2	4	4	2
E000002	3	2	1	0	0	0	0	0	1	1	1	0	0
E000003	1	3	3	1	1	1	1	1	3	2	3	0	0

- What we want

	Q1a	Q1b	Q1c	Q2a	…
0 marks	30	120	32	153	…
1 mark	118	132	64	43	…
2 marks	132	142	32	163	…

Pivoting for Stacked Bar

```
head = rows[0][1:]
table = [[0] * len(head) for i in range(6)]  # max 5 marks

for row in rows[1:]:
    for qn, marks in enumerate(row[1:]):
        table[marks][qn] += 1

btm = [0] * len(head)
for mark, row in enumerate(table):
     ax.bar(head, row, bottom=btm,
            label=mark)
     btm = [x+y for x,y in zip(btm, row)]
```

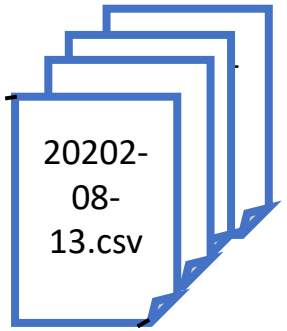


Plotting Time Series

- One of the axis is time
  - usually x-axis
- Use `mathplotlib.dates.datestr2num`
  - converts common date formats to matplotlib number
- Plot with `ax.plot_date(x, y)`

Coursemology EXP

- Daily snapshots of students statistics
  - Into csv files named: YYYY-MM-DD.csv
- CSV structure:



Student ID	Name	Level	EXP
47562	xxx	1	75
…	…	…	…

- What we need:

Key	Value
47562	
47693	
…	

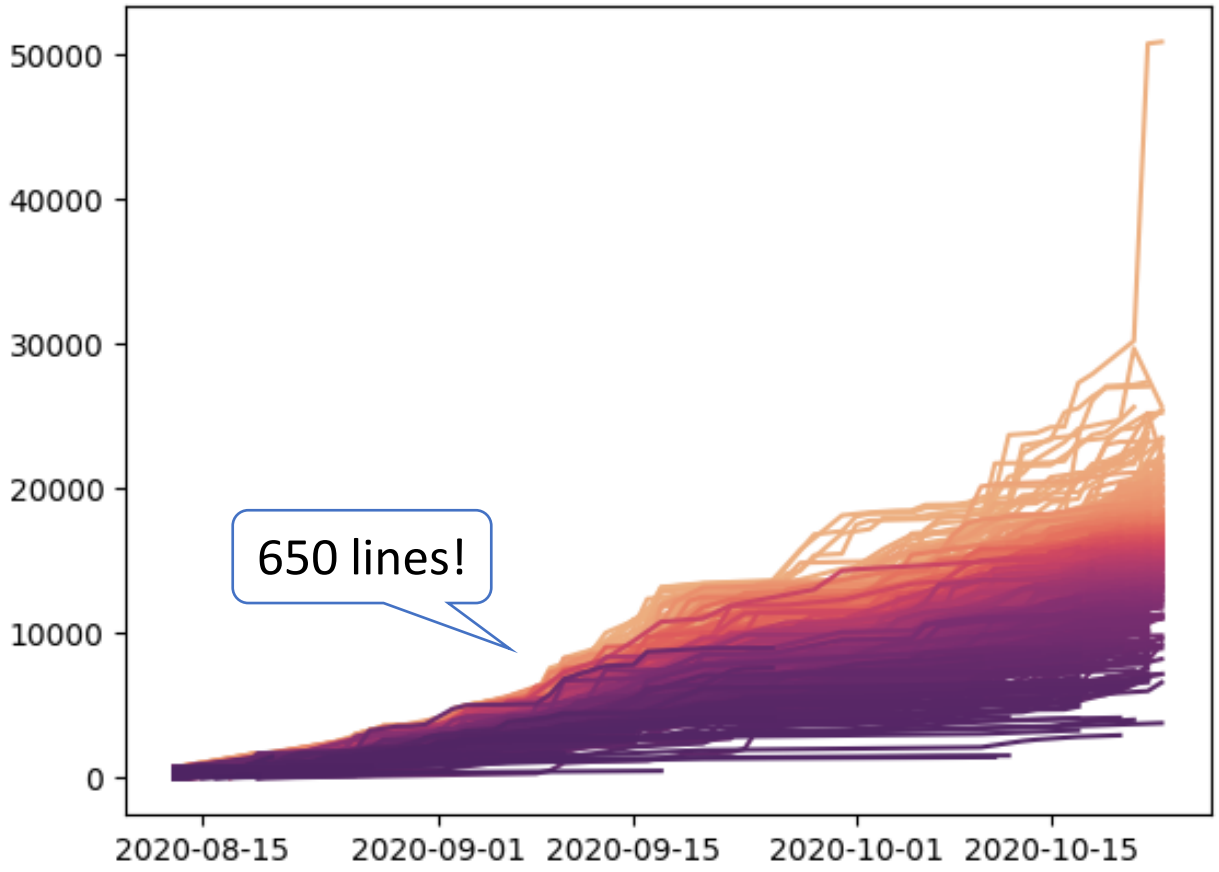
Date	13/8	17/8	18/8	…
EXP	75	250	450	…

Concatenate and Pivot

```
from glob import glob
from matplotlib.dates import datestr2num

table = {}

for file in sorted(glob('*.csv')):
    rows = read_csv(file)
    date = datestr2num(file[:-4])
    for row in rows[1:]:
        sid, exp = row[0], int(row[3])
        if sid not in table:
            table[sid] = [(date, exp)]
        elif table[sid][-1][1] != exp:
            table[sid].append((date, exp))
```

Plot each student

```
fig, ax = plt.subplots()

for row in table.values():
    x, y = zip(*row)
    ax.plot_date(x, y, '-',
    color=sns.color_palette…
fig.show()
```



Another real-world dataset

- COVID19 Data Repository
  - By CSSE at John Hopkins
- Time-series data
  - Confirmed
  - Deaths
  - Recovered
- Compiled in wide-form
  - From daily reports

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	…

https://github.com/CSSEGISandData/COVID-19

Examining the dataset

- Don’t open with Excel
  - You have been warned
- Use a proper editor
  - Notepad++
  - Visual Studio Code (with edit csv extension)
- Examine “oddities”
  - Province/State can be blank
  - Country/Region contains duplicates

What shall we plot?

- Merge (sum) countries together
  - Not interested in individual states

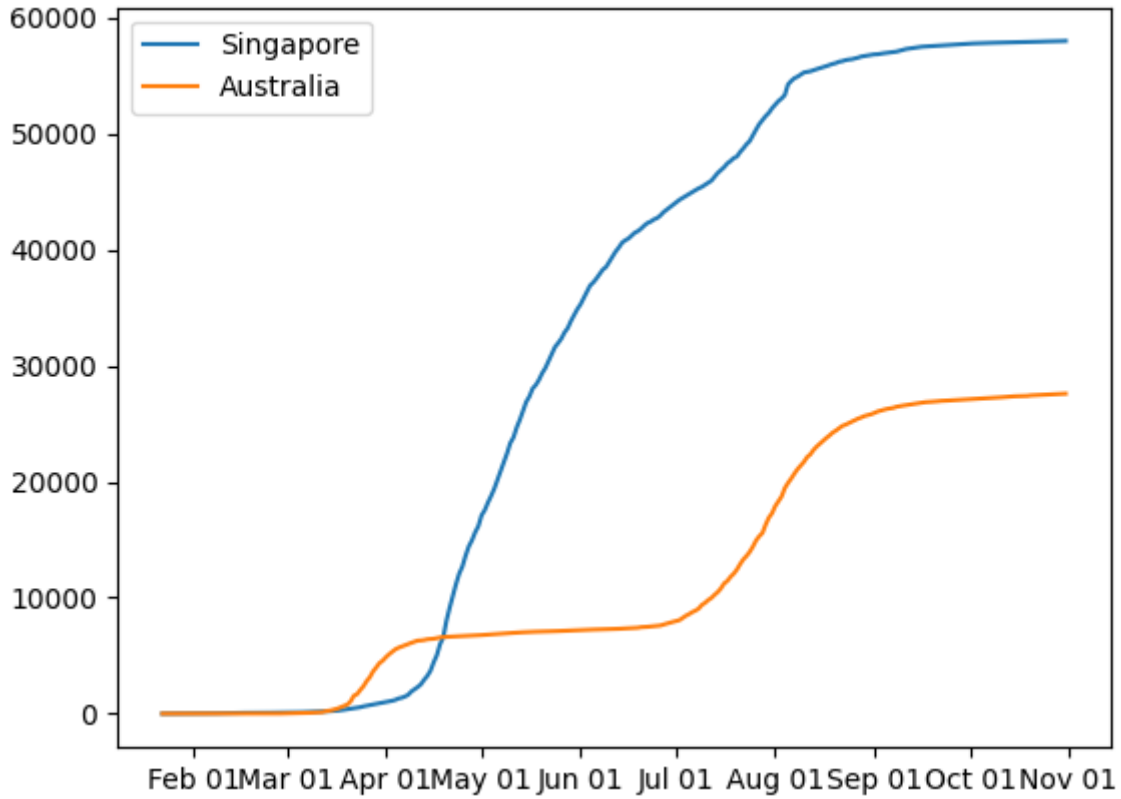
- Total cases over time
  - Most basic plot

Total cases over time

```
data = read_csv("time_series_covid19_confirmed_global.csv")
xtics = tuple(map(dates.datestr2num, data[0][4:]))

# sum by country
table = {}
for row in data[1:]:
    c, vals = row[1], [int(x) for x in row[4:]]
    if c not in table:
        table[c] = vals
    else:
        for i in range(len(vals)):
            table[c][i] += vals[i]
```

Total cases over time

```
countries = ['Singapore', 'Australia', 'Korea, South']
for c in countires:
    ax.plot_dates(xtics, table[c], label=c)

ax.legend()
fig.show()
```



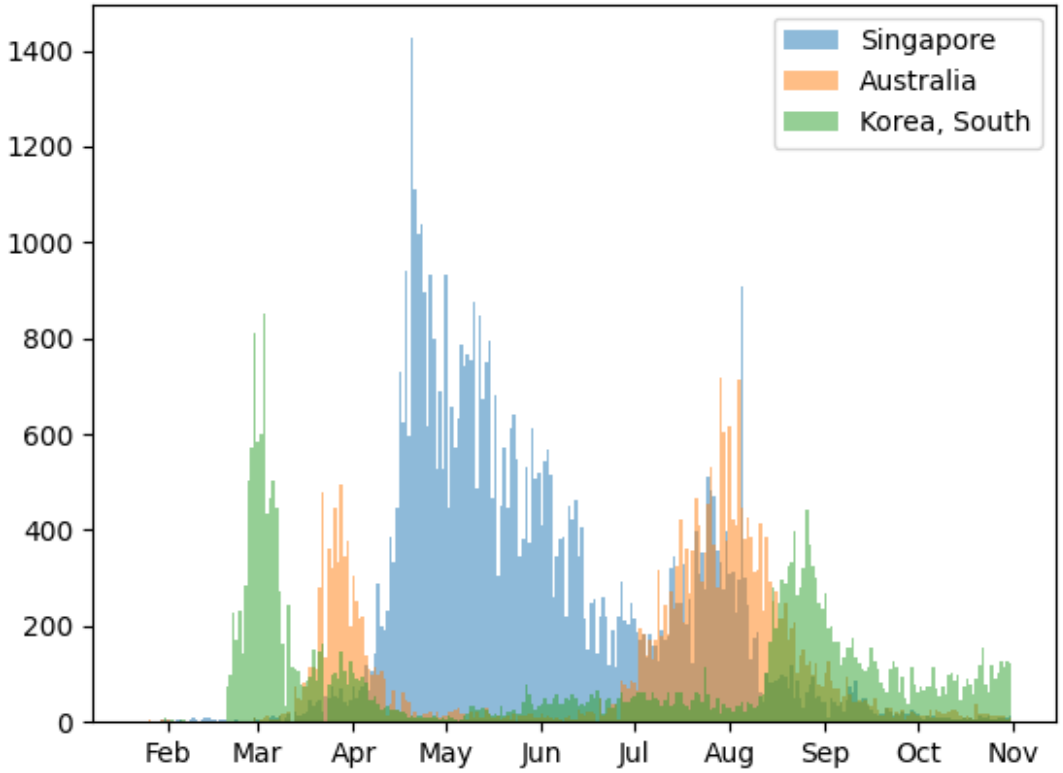
What shall we plot?

- Merge (sum) countries together
  - Not interested in individual states

- Total cases over time
  - Most basic plot
- New cases daily
  - Change in total cases per day

New Cases Daily

```
def delta(row):
    ret = [row[0]]
    for i in range(len(row)-1):
        ret.append(row[i+1] - row[i])
    return ret

for c in countries:
    ax.bar(xtics, delta(table[c]))
ax.xaxis_date()
```



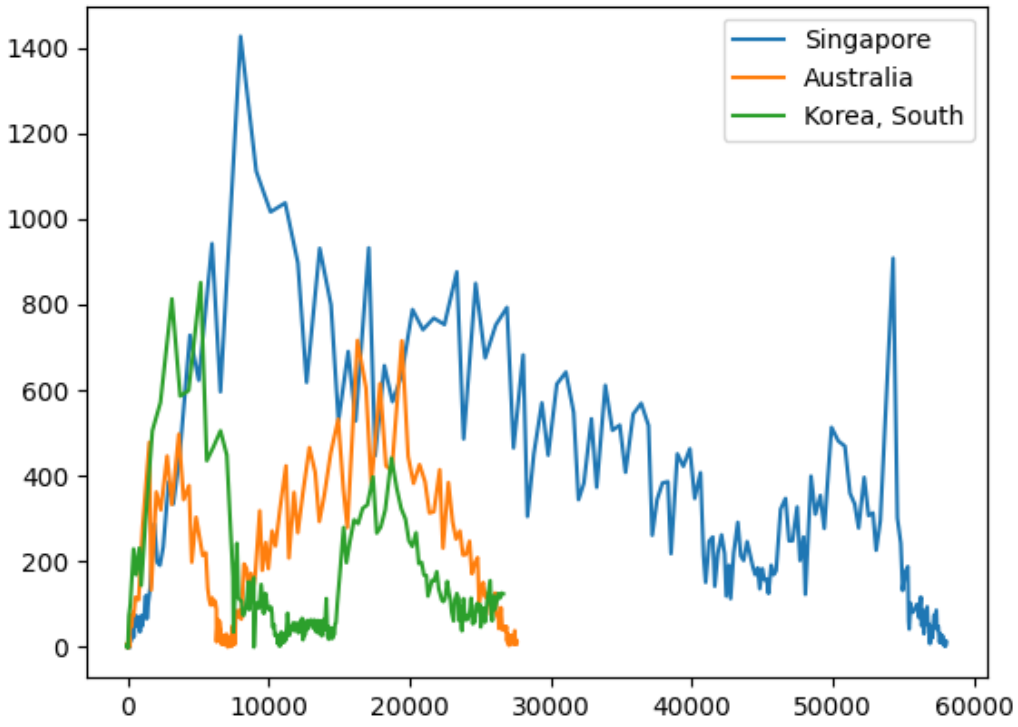
Are these plots good

- Depends on what we want to know
- Compare between countries
  - Need to align x-axis (time domain)
  - Rate of growth (how exponential)
- Normalize other parameters
  - Population, existing cases

Time is irrelevant

- Exponential growth
  - increase by a constant factor > 1
  - does not depend on time
  - double every 5 days v.s. double every 10 days
- Plot against total cases

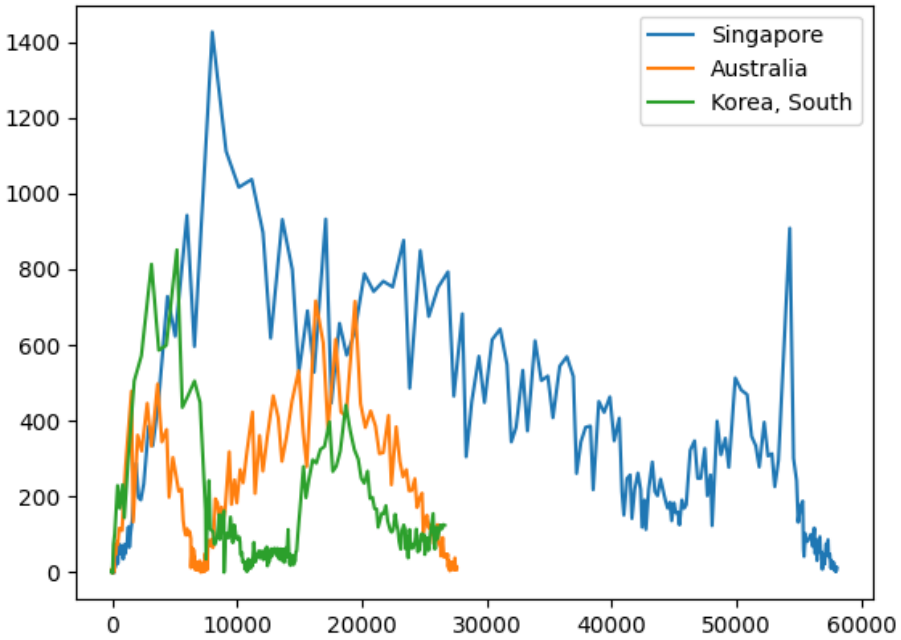
```
ax.plot(table[c], delta(table[c]))
```

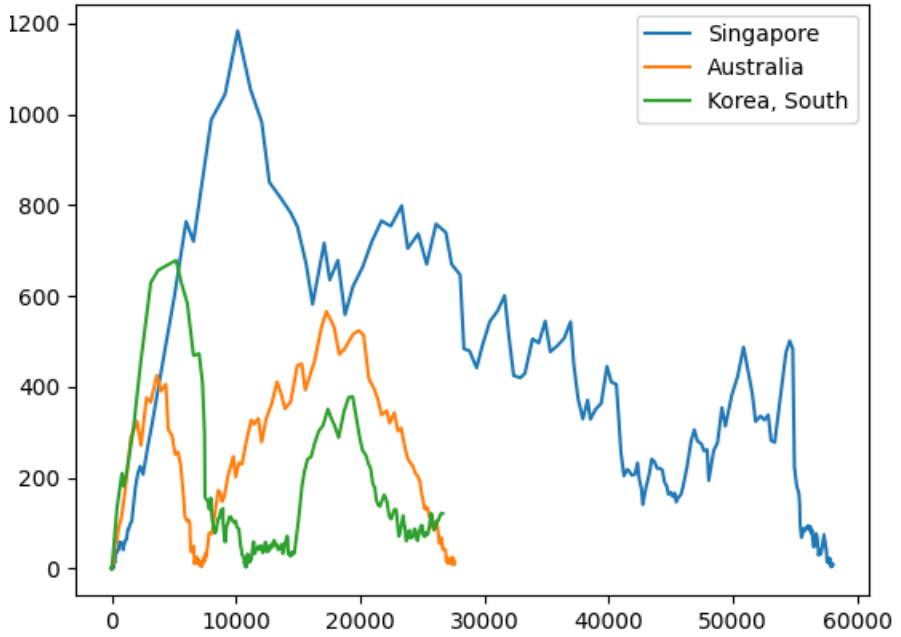


Smoothing

- Daily new cases too “choppy”
  - Moving average

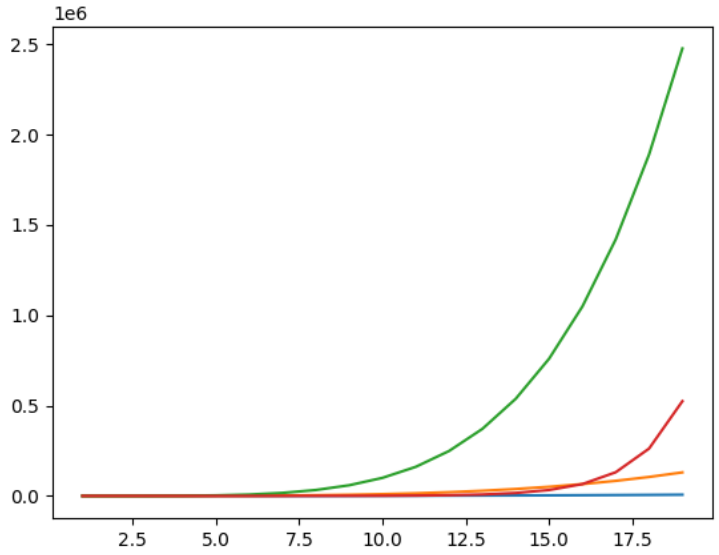
```
def mavg(row, n):
    return [sum(row[max(0, i-n):i])/min(i, n) \
            for i in range(1, len(row)+1)]
```

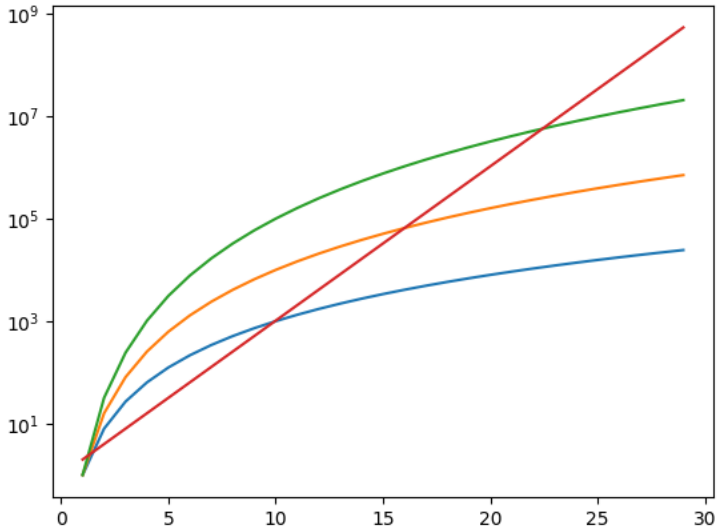


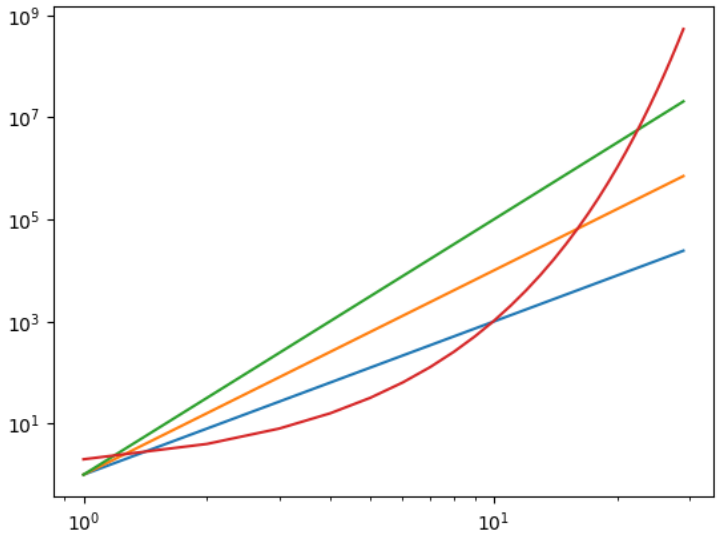


Axis Scale

- Which line is exponential growth?

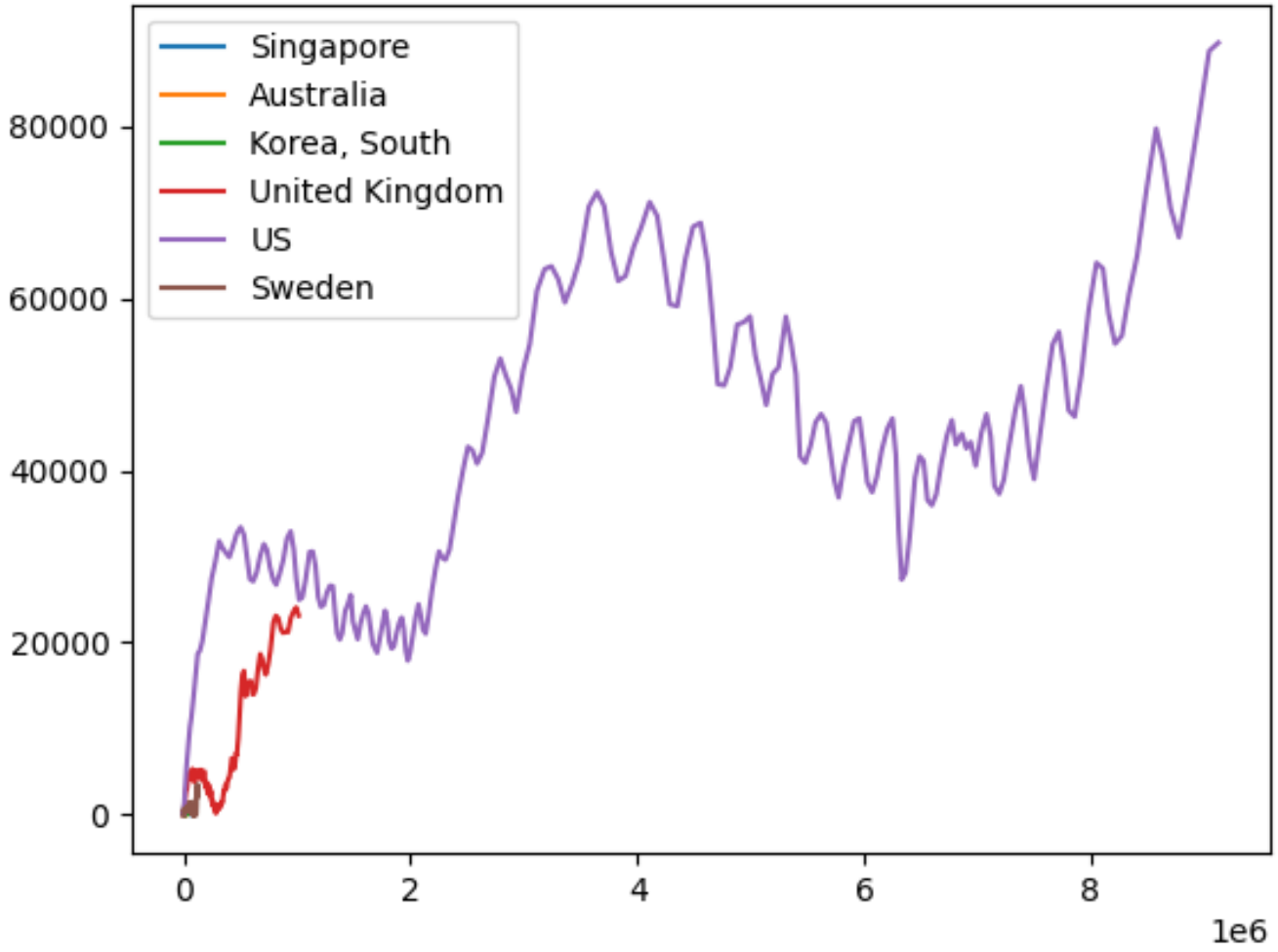






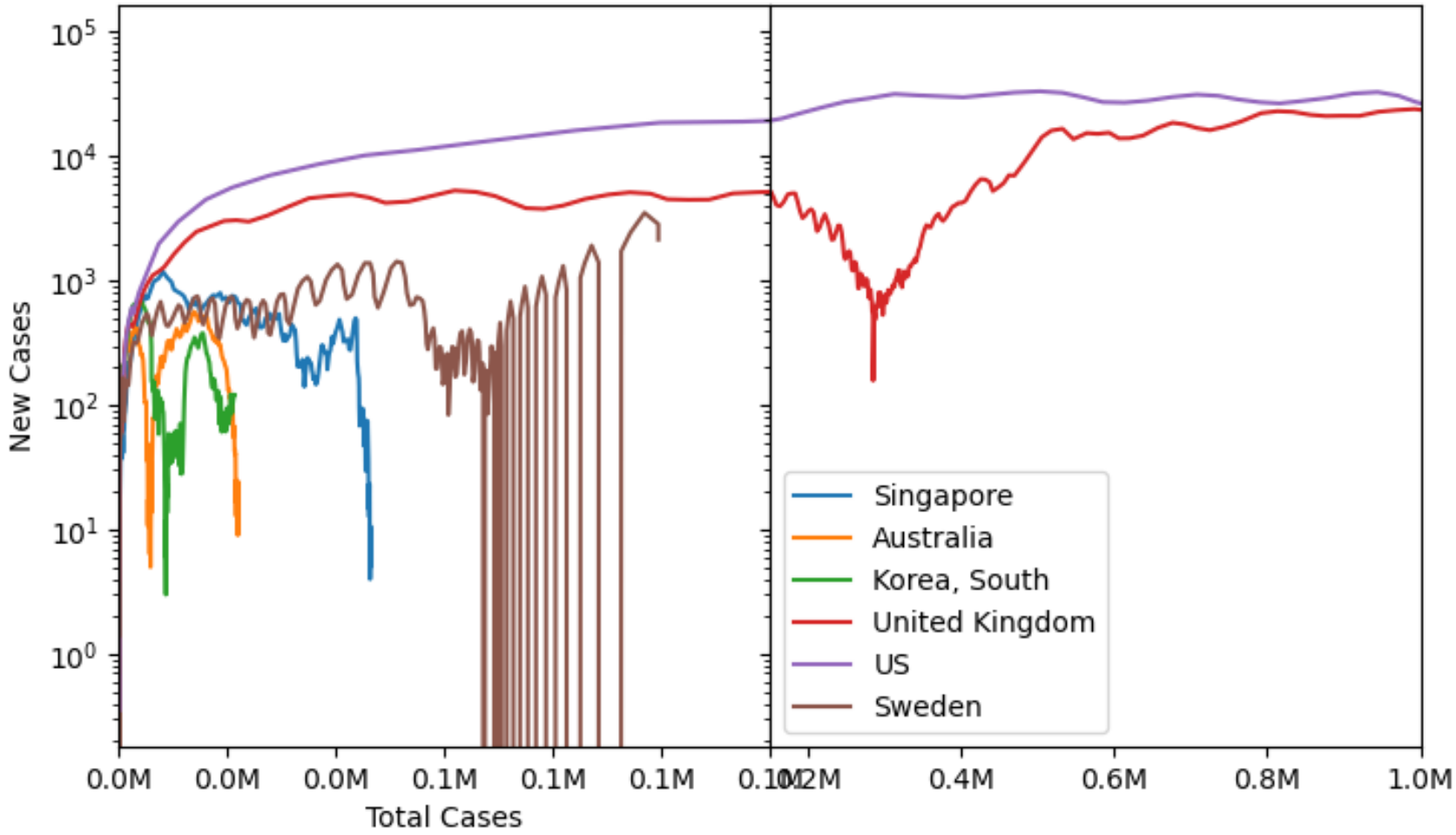
Linear-Linear

- Differences in plots are exponentially large



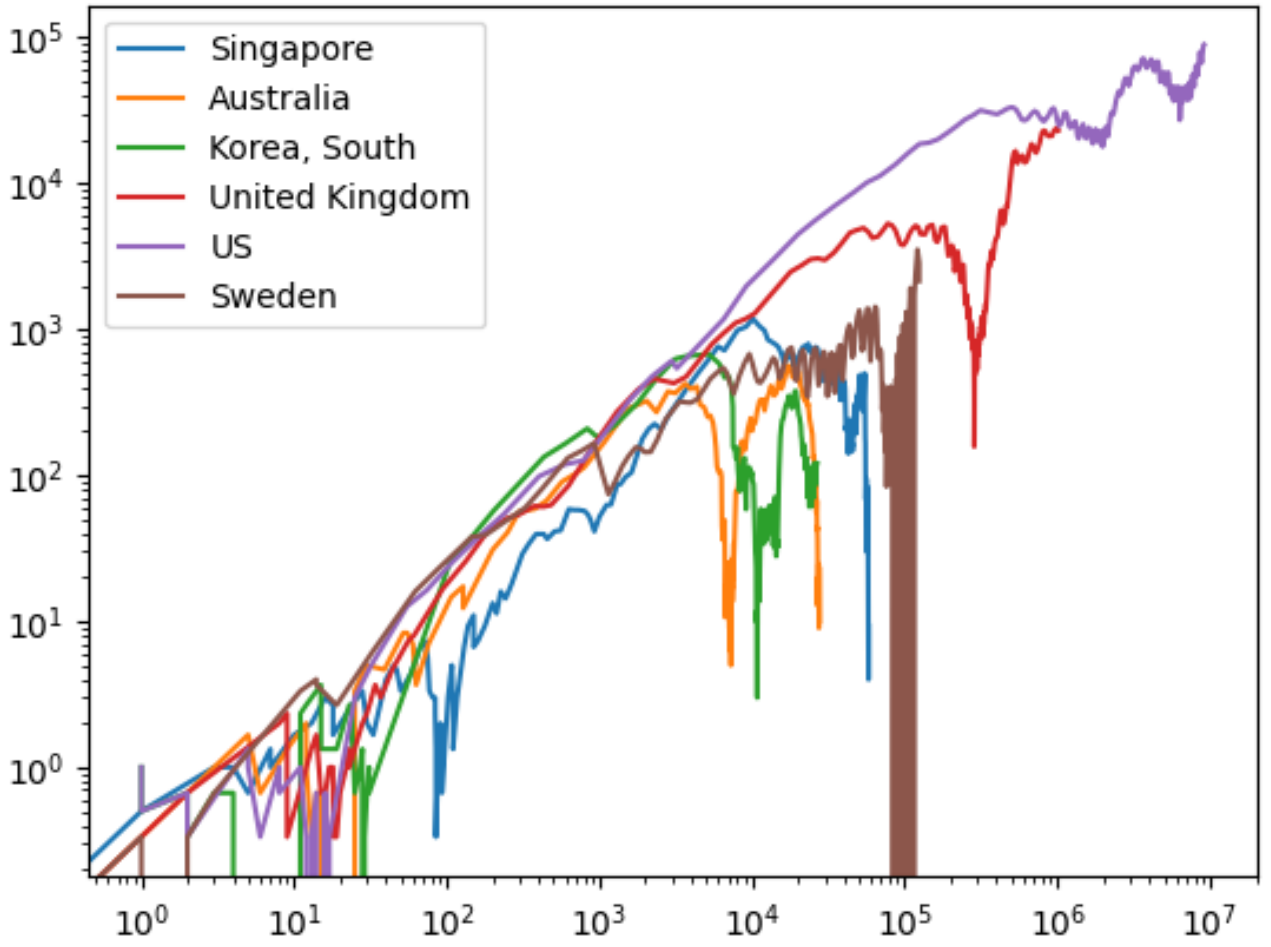
Log-Linear

- X-axis is split into two scales, but both are linear

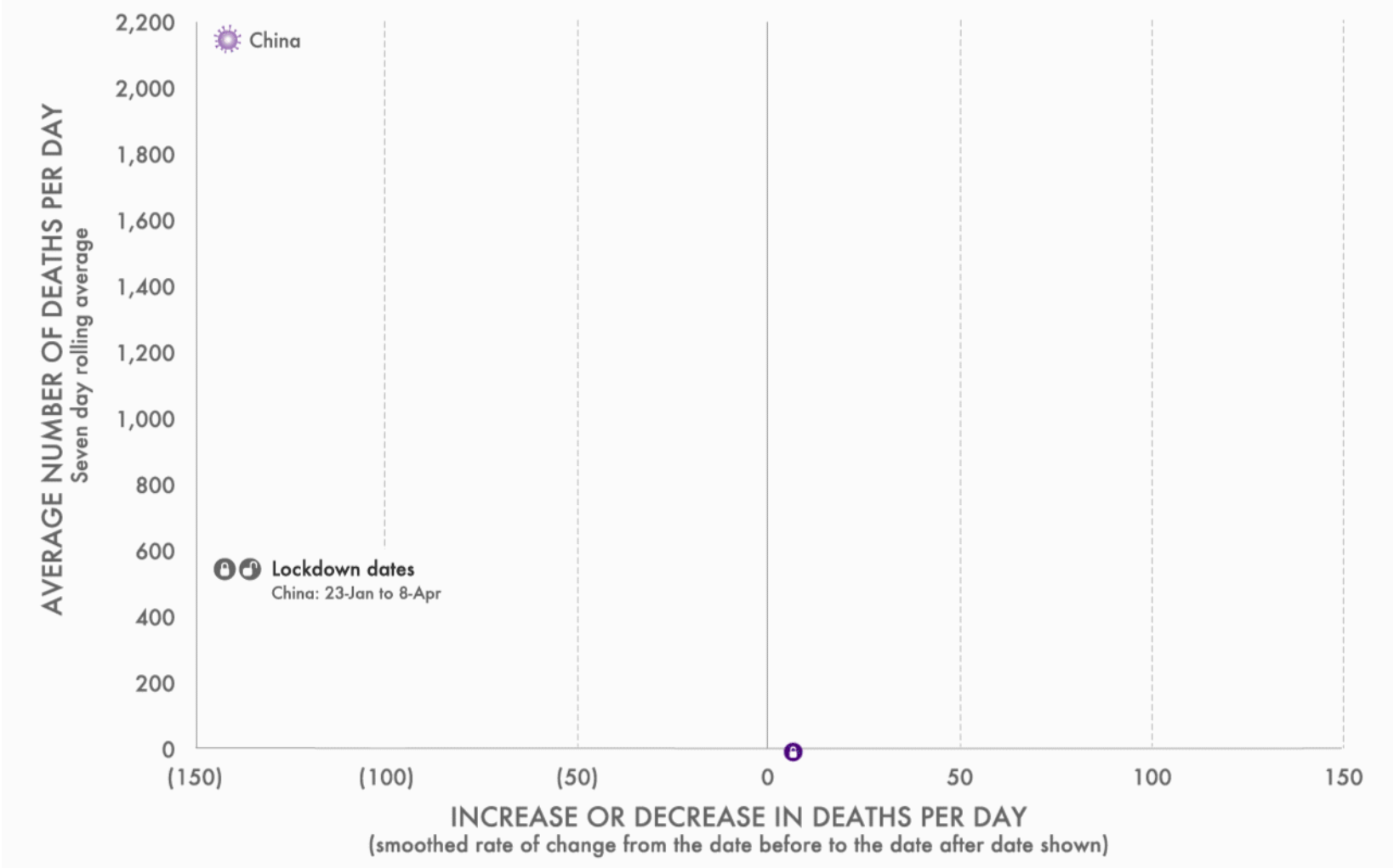


Log-Log

- Seems polynomial growth overall



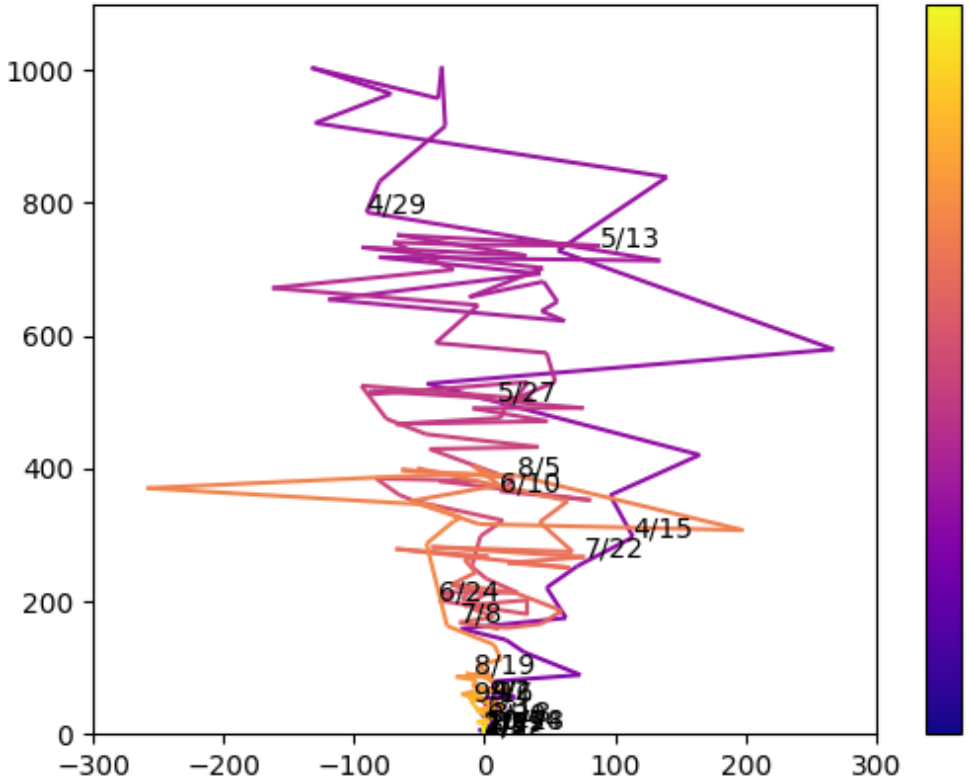
Yet another way to plot



https://theconversation.com/three-graphs-that-show-a-global-slowdown-in-covid-19-deaths-135756

Yet another way to plot

- Y-axis: Average number of cases per day
  - `mavg(delta(table[c]), 7)`
- X-axis: Increase of decrease in new cases
  - `mvag(delta(delta(table[c])), 3)`



A bunch of useful functions

- delta
- mavg
- adding values of lists together
- etc.

The two most common libraries

- Powerful n-dimensional array (list) abstraction
  - Basically lists and list-of-lists, list-of-list-of-lists, etc.
- Lots of functions implemented
  - Random generators, linear algebra, Fourier transforms
- Works like Python lists
  - But with better performance
  - Core written in C code

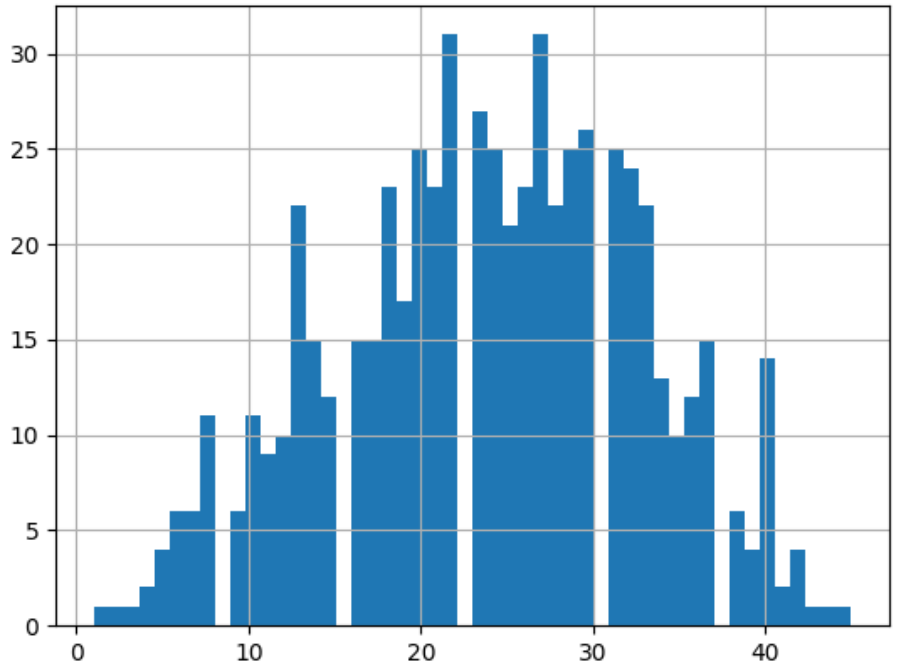
- DataFrame
  - Table abstraction
  - Dictionary-of-dictionaries
- Reads csv directly
  - Automatic header and type detection
- Advance functions
  - Group-by, filter, reshaping, pivoting
- Can plot without Matplotlib
  - Uses Matplotlib internally

Midterm Marks Revisited

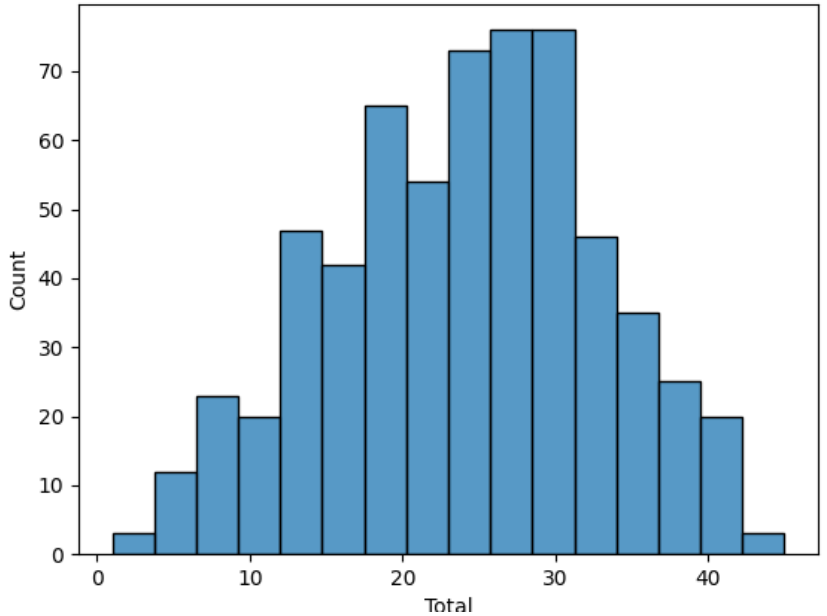
- Source data
  - need total column

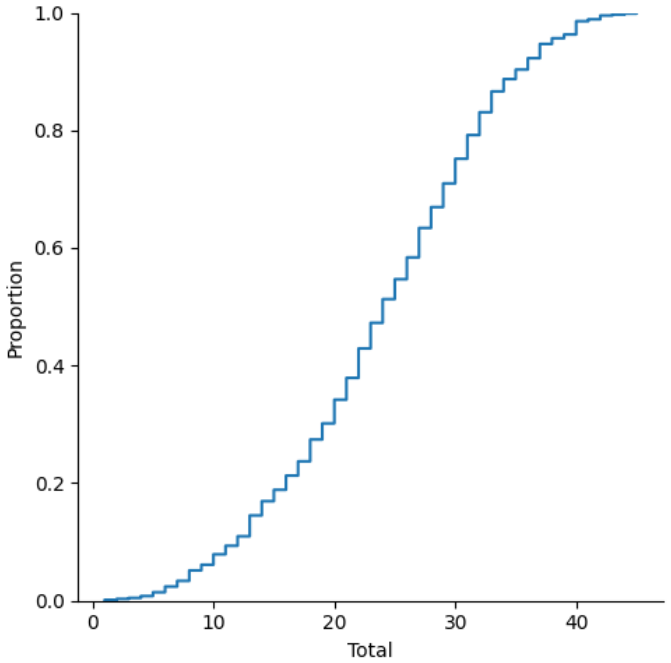
ID	Q1a	Q1b	Q1c	Q2a	Q2b	Q2c	Q2d	Q3	Q4a	Q4b	Q4c	Q4d	Q4e
E000001	5	1	5	5	2	5	2	3	3	2	4	4	2
E000002	3	2	1	0	0	0	0	0	1	1	1	0	0
E000003	1	3	3	1	1	1	1	1	3	2	3	0	0

```
import pandas as pd

df = pd.read_csv("midterm-marks.csv")
df['Total'] = df.sum(axis=1)

df['Total'].hist(bins=50)
```




```
import seaborn as sns

sns.histplot(data=df, x='Total')

sns.displot(data=df, x='Total’,
            kind='ecdf')
```



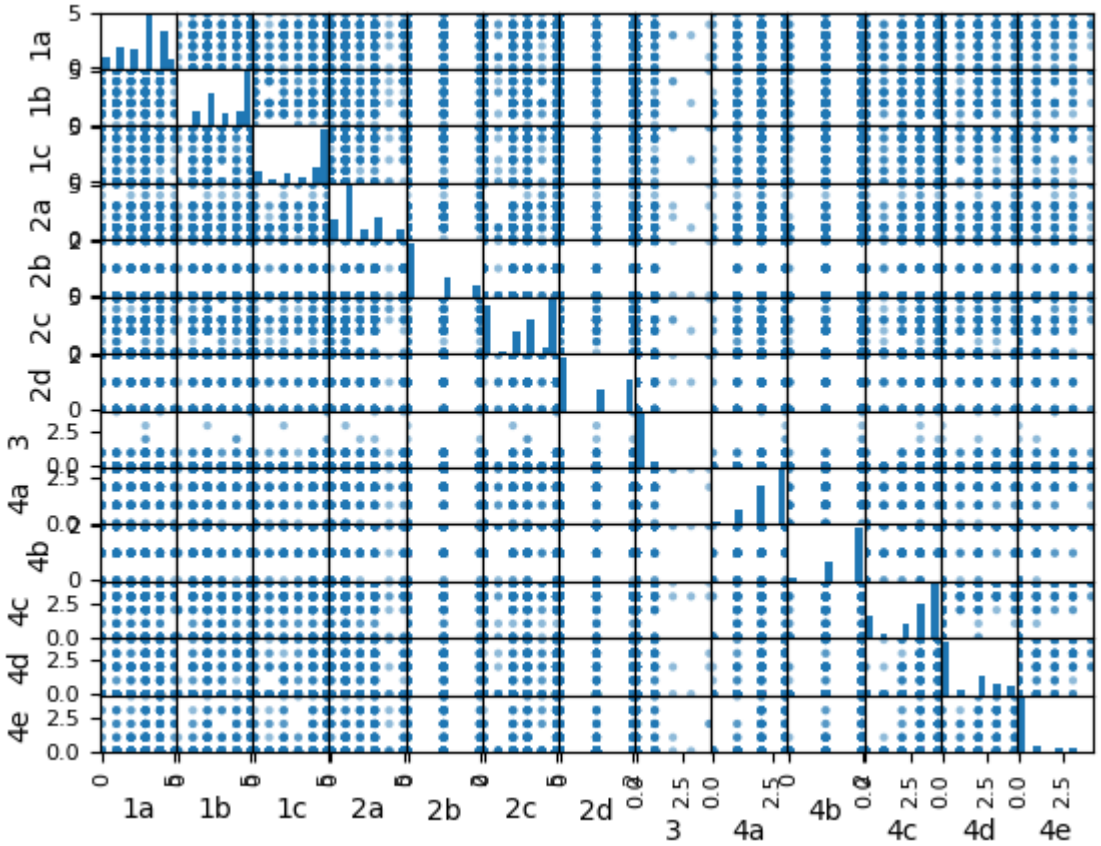


One last example

- Corelation analysis for midterm marks
  - between questions

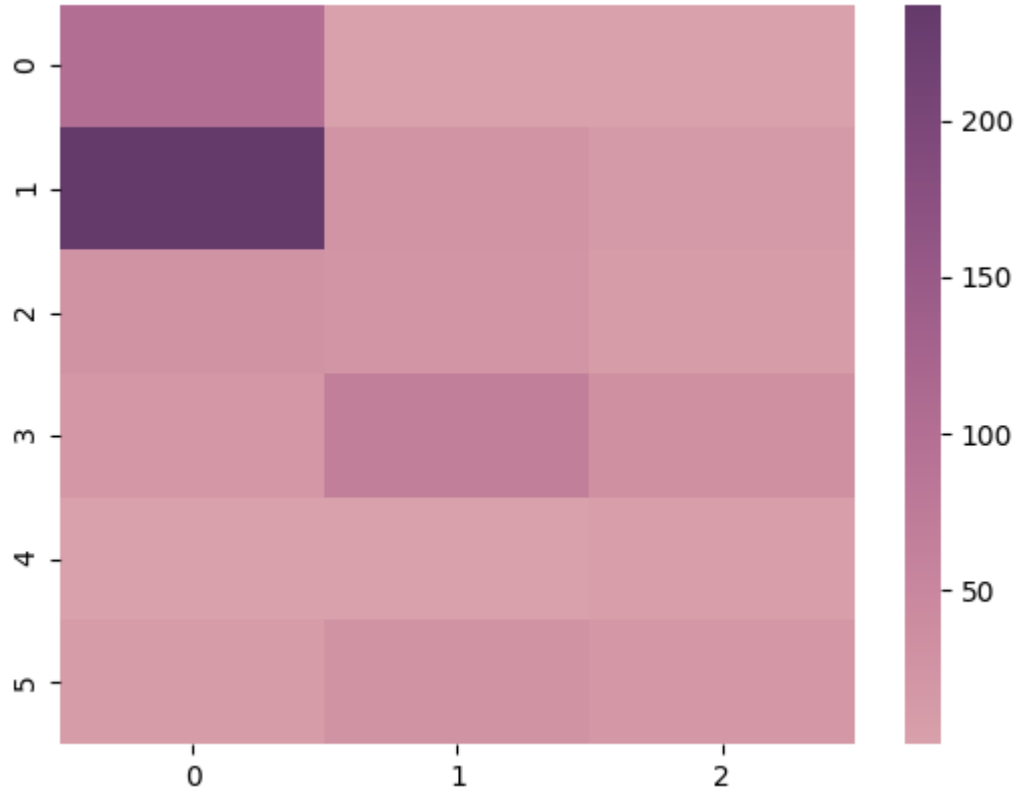
    `pd.plotting.scatter_matrix(df)`

- We can do better
  - some amount of manual work



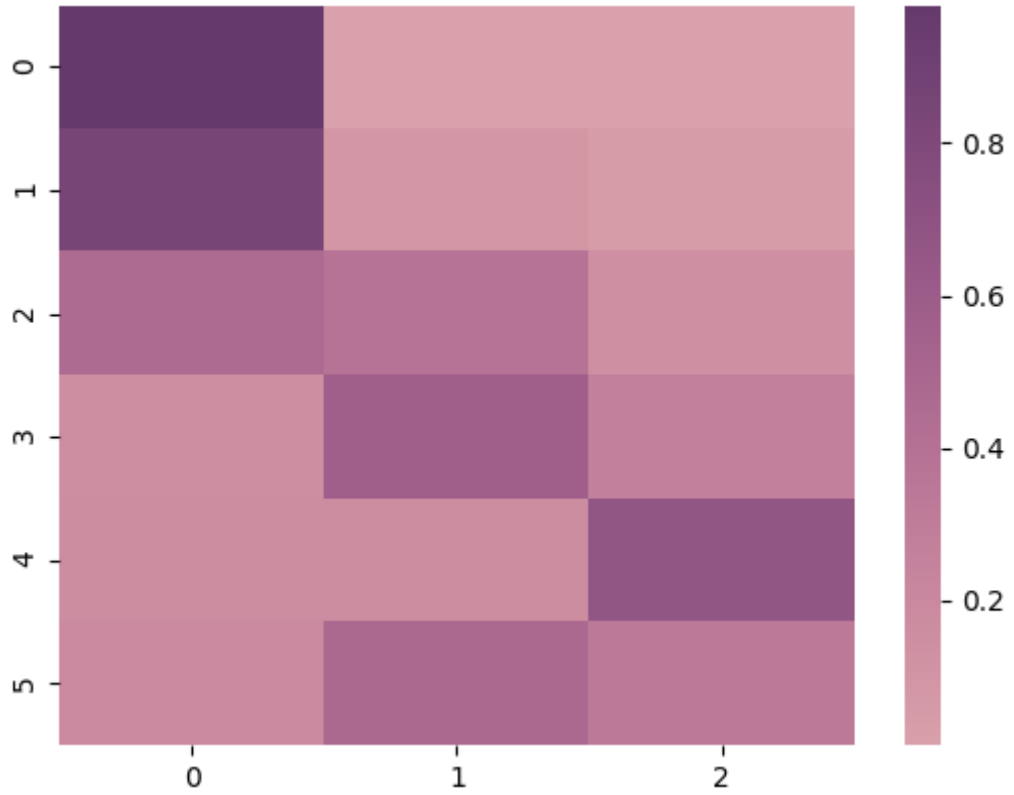
Building a heatmap

```
>>> hm, x, y = np.histogram2d(df['2a'], df['2b’],
                              bins=[range(7), range(4)])
>>> hm
array([[102.,   1.,   1.],
       [237.,  26.,  13.],
       [ 27.,  23.,   9.],
       [ 18.,  67.,  32.],
       [  1.,   1.,   4.],
       [ 11.,  28.,  19.]])
>>> x, y
(array([0, 1, 2, 3, 4, 5, 6]),
 array([0, 1, 2, 3]))

>>> sns.heatmap(hm)
```



Scale along rows

```
>>> (hm.T/hm.sum(axis=1)).T
array([[0.98, 0.01, 0.01],
       [0.86, 0.09, 0.05],
       [0.46, 0.39, 0.15],
       [0.15, 0.57, 0.27],
       [0.17, 0.17, 0.67],
       [0.19, 0.48, 0.33]])
```

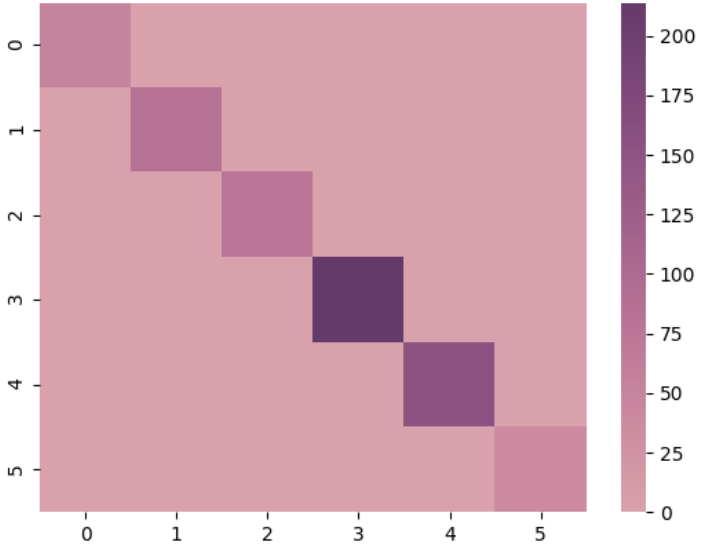


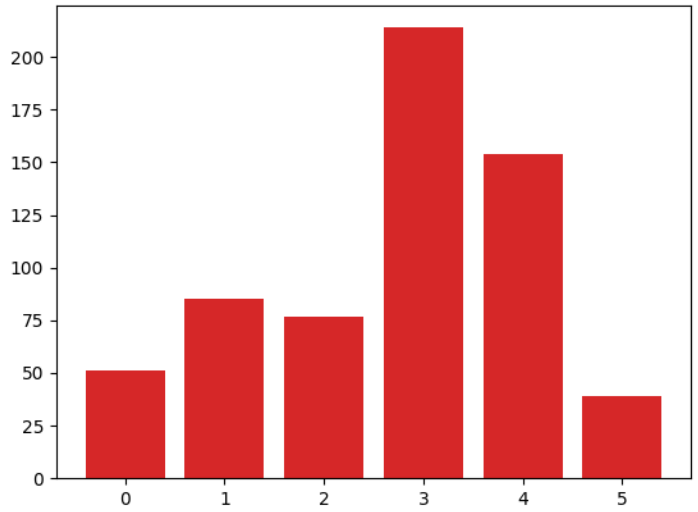
Build a 2D Table

```
marks = {'1a': 5, '1b': 5, '1c': 5,
         '2a': 5, '2b': 2, '2c': 5, '2d': 2,
         '3': 4,
         '4a': 3, '4b': 2, '4c': 4, '4d': 4, '4e': 4}
table = {}
for i, v in marks.items():
    table[i] = {}
    for j, w in marks.items():
        hm, x, y = np.histogram2d(df[i], df[j],
                                  bins=[range(v+2), range(w+2)])
        table[i][j] = hm if i == j else (hm.T/hm.sum(axis=1)).T
```

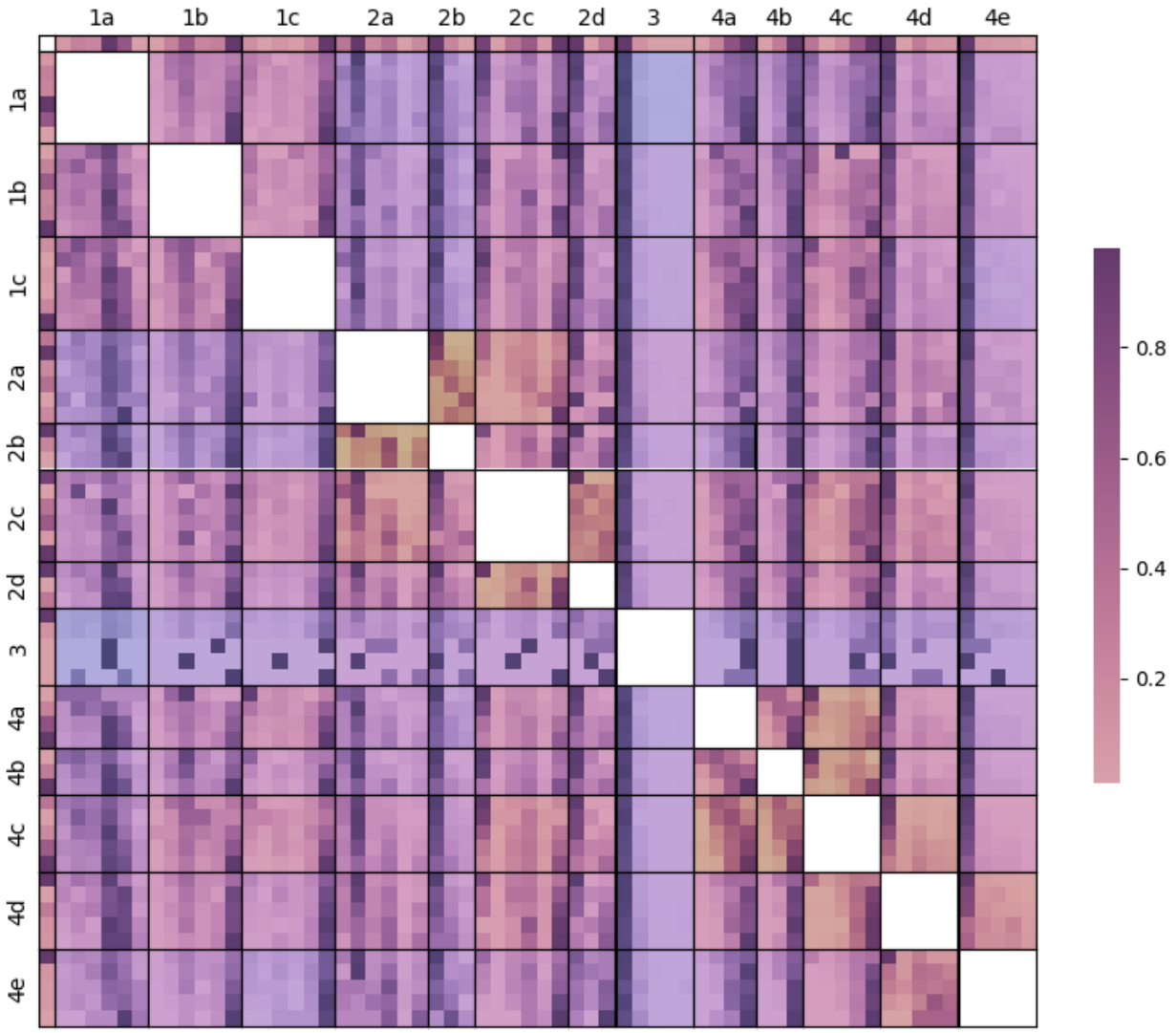
The diagonals

```
>>> table['1a']['1a']
array([[ 51.,   0.,   0.,   0.,   0.,   0.],
       [  0.,  85.,   0.,   0.,   0.,   0.],
       [  0.,   0.,  77.,   0.,   0.,   0.],
       [  0.,   0.,   0., 214.,   0.,   0.],
       [  0.,   0.,   0.,   0., 154.,   0.],
       [  0.,   0.,   0.,   0.,   0.,  39.]])

>>> table['1a']['1a'].diagonal()
array([51.,  85.,  77., 214., 154.,  39.])
```



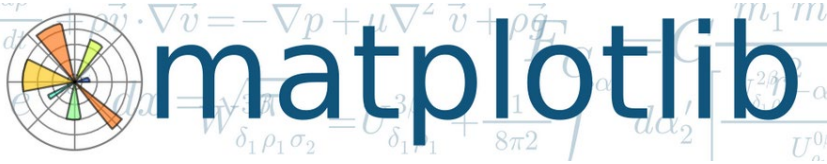


A bit of layout



Summary

- Data visualisation
  - Manipulating data
  - Pivoting, counting, etc
- Libraries makes standard operations easy
  - But cannot use in PE
  - Learn the fundamentals



What’s next

- We only scratched the surface of visualisation
  - https://matplotlib.org/3.1.0/gallery/index.html
  - https://seaborn.pydata.org/examples/index.html
  - https://www.data-to-viz.com/
- Other types of plots
  - Word clouds, Graphs, maps
  - Sankey, Venn diagrams
- Additional features
  - Animation
  - Interactive plots