

# YData: Introduction to Data Science



Class 10: Intro to data visualization

# Overview

Quick review of pandas DataFrames

Joining DataFrames

History of Data Visualization

If there is time matplotlib:

- Line graphs
- Histograms



# Announcement: Homework 4

Homework 4 has been posted!

It is due on Gradescope on **Sunday February 19<sup>th</sup> at 11pm**

- **Be sure to mark each question on Gradescope along with the **page that has the answers!****

Note: On question 1.4 it should say:

To start, please extract a DataFrame that has just two columns:

- **Primary Fur Color**: The main fur color of each squirrel (Not "**The Primary Fur Color**")
- Indifferent : Whether the squirrel was indifferent



# Series and Tables

# Review: pandas DataFrames

PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter..."	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

Pandas DataFrame hold Table data

Selecting columns:

- `my_df[["col1", "col2"]].copy()` # getting multiple columns using a list

Extracting rows:

- `my_df.iloc[0]` # getting a row by number
- `my_df.loc["index_name"]` # getting a row by Index value
- `my_df[my_df["col_name"] == 7]` # getting rows using a Boolean mask

# Review: pandas DataFrames

## Sorting rows of a DataFrame

```
my_df.sort_values("col_name", ascending = False)  # sort from largest to smallest
```

## Adding a new:

- `my_df["new_col"] = values_array`

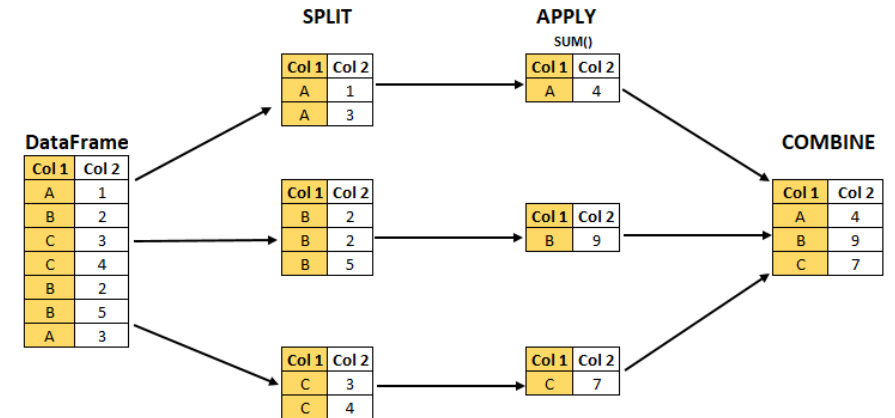
## Renaming a column:

- `rename_dictionary = {"old_col_name": "new_col_name"}`
- `my_df.rename(columns = rename_dictionary )`

# Creating aggregate statistics by group

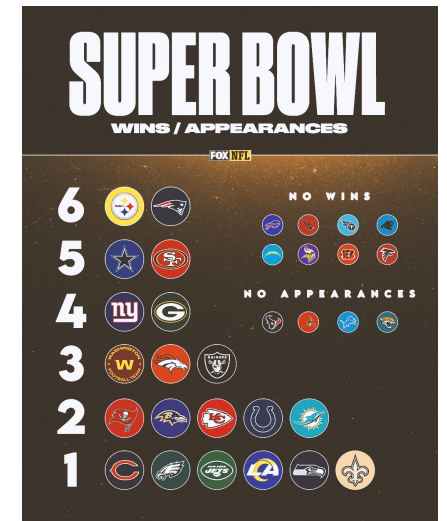
We can get statistics separately by group:

- `dow.groupby("Year").agg("max")`

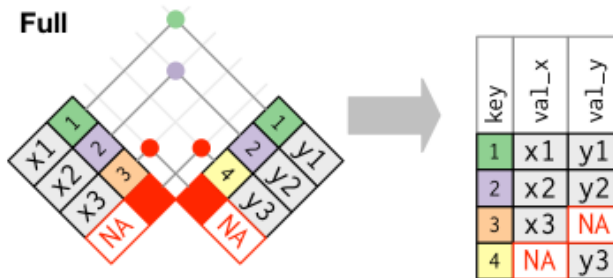
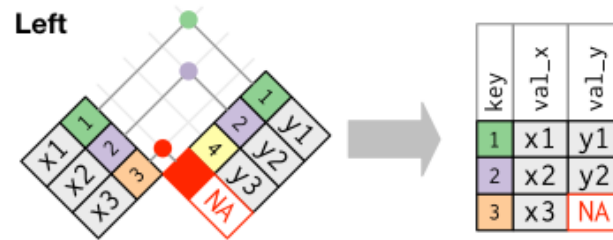


```
my_df.groupby("group_col_name").agg(  
    new_col1 = ('col_name', 'statistic_name1'),  
    new_col2 = ('col_name', 'statistic_name2'),  
    new_col3 = ('col_name', 'statistic_name3')  
)
```

Let's do a few more practice problems in Jupyter!



# Joining data frames

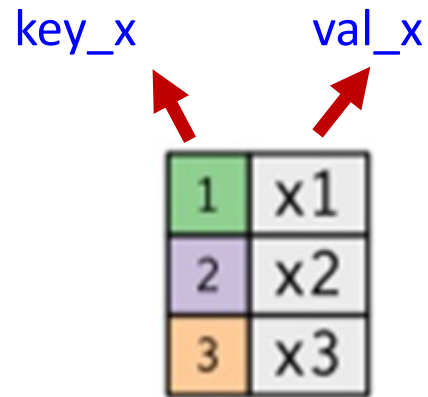




# Left and right tables

Suppose we have two DataFrames (or Series) called **x\_df** and **y\_df**

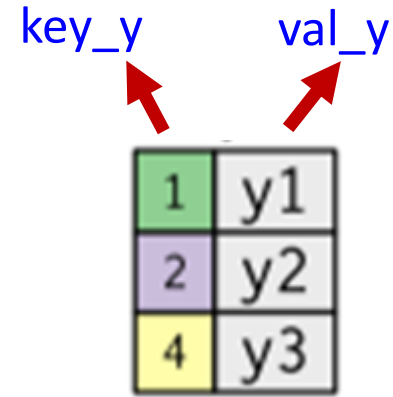
- **x\_df** have two columns called **key\_x**, and **val\_x**
- **y\_df** has two columns called **key\_y** and **val\_y**



A 3x2 grid representing a DataFrame. The first column contains values 1, 2, and 3, each in a colored box (green, purple, orange respectively). The second column contains values x1, x2, and x3 in grey boxes. Red arrows point from the labels 'key\_x' and 'val\_x' to the first and second columns respectively.

1	x1
2	x2
3	x3

**DataFame: x\_df**



A 3x2 grid representing a DataFrame. The first column contains values 1, 2, and 4, each in a colored box (green, purple, yellow respectively). The second column contains values y1, y2, and y3 in grey boxes. Red arrows point from the labels 'key\_y' and 'val\_y' to the first and second columns respectively.

1	y1
2	y2
4	y3

**DataFrame y\_df**

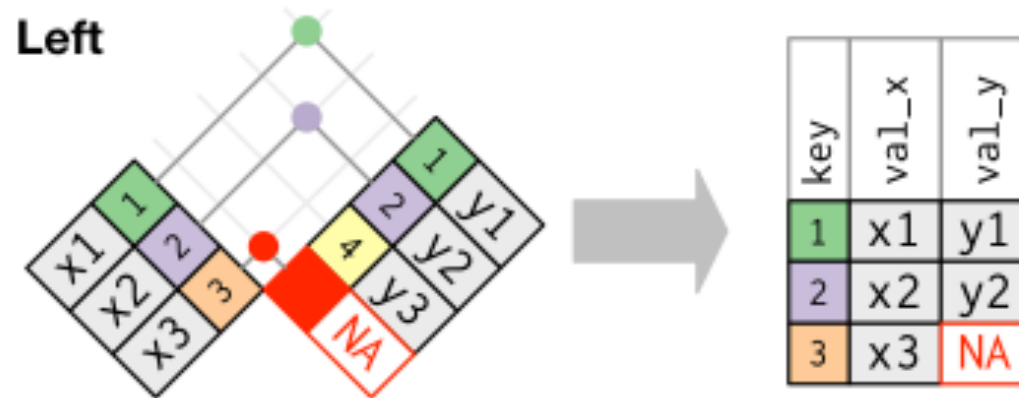
Joins have the general form:

```
x_df.merge(y_df, left_on = "key_x", right_on = "key_y")
```

# Left joins

**Left joins** keep all rows in the left table.

Data from right table is added when there is a matching key, otherwise NA is added.

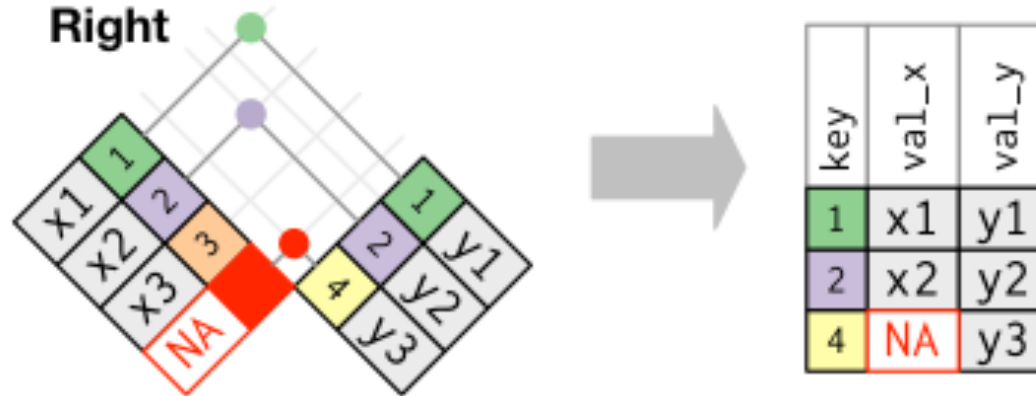


```
x_df.merge(y_df, how = "left", left_on = "key_x", right_on = "key_y")
```

# Right joins

**Right joins** keep all rows in the right table.

Data from left table added when there is a matching key, otherwise NA is added.



```
x_df.merge(y_df, how = "right", left_on = "key_x", right_on = "key_y")
```

# Inner joins

**Inner joins** only keep rows in which there are matches between the keys in both tables.

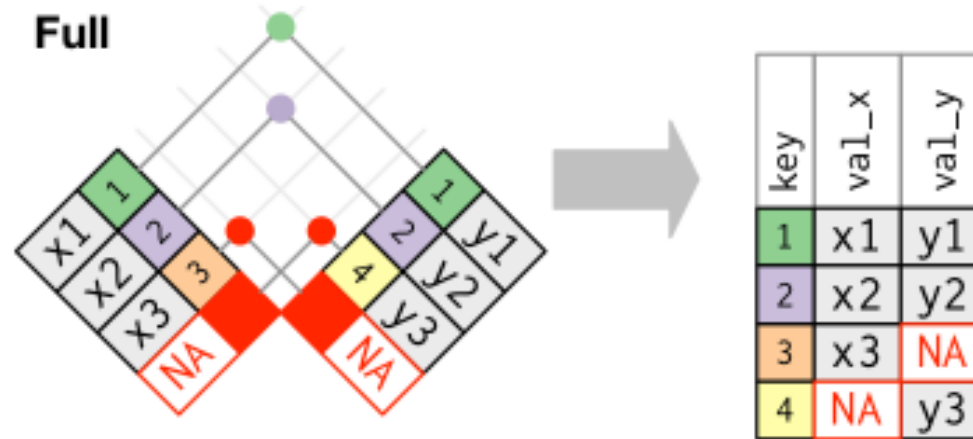


```
x_df.merge(y_df, how = "inner", left_on = "key_x", right_on = "key_y")
```

# Full (outer) joins

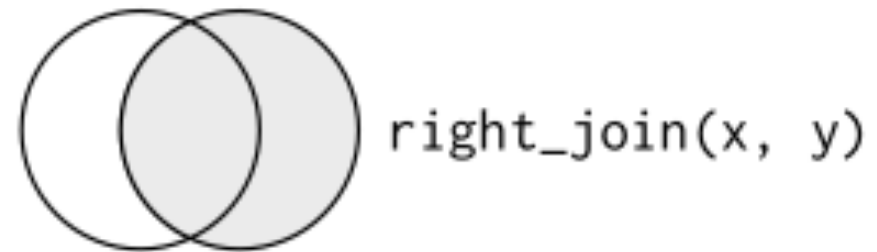
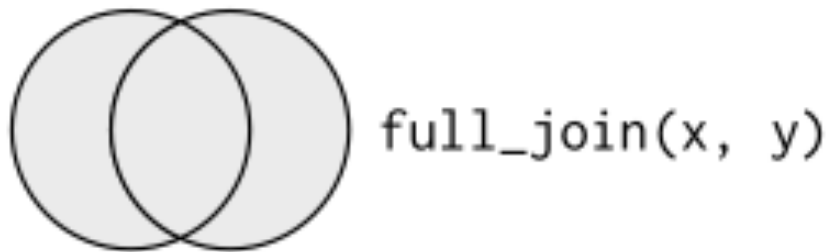
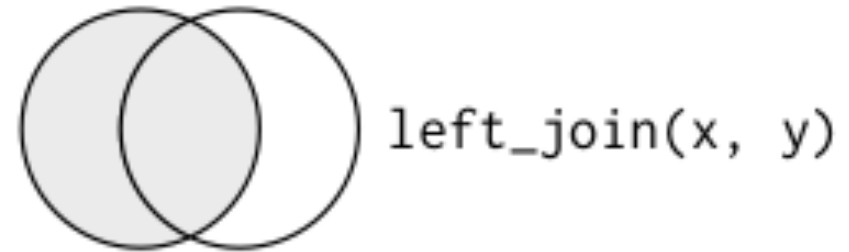
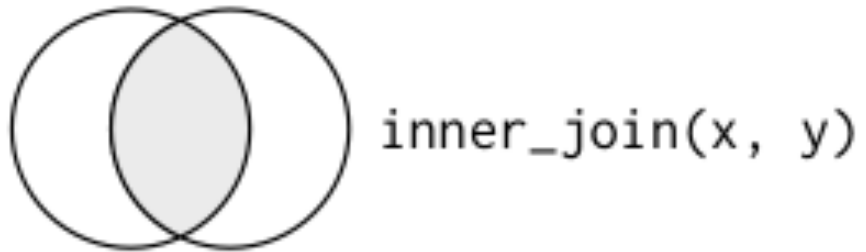
**Full joins** keep all rows in both table.

NAs are added where there are no matches.



```
x_df.merge(y_df, how = "outer", left_on = "key_x", right_on = "key_y")
```

# Summary



# Joining on Index values

If two DataFrames **have the same Index values** then we can join them using the `.join()` method instead of the `.merge()` method

The `.join()` method is very similar to `.merge()` except we don't need to specify `left_on` and `right_on` arguments since the DataFrames are being joined by their Indexes

An example of a left join would be:

- `x_df.join(y_df, how = "left")` # assuming x\_df and y\_df has the same Index values

Let's explore this in Jupyter!

Questions?





# Data visualization!



# A very brief history of data visualization...

*Statistical Science*  
2008, Vol. 23, No. 4, 502–535  
DOI: 10.1214/08-STS268  
© Institute of Mathematical Statistics, 2008

## **The Golden Age of Statistical Graphics**

**Michael Friendly**

# Data visualization



What are some reasons we visualize data rather than just reporting statistics?

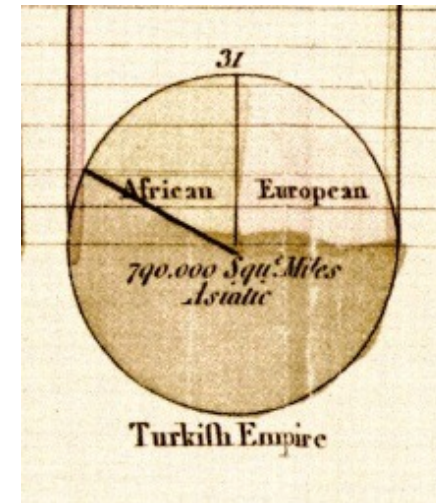
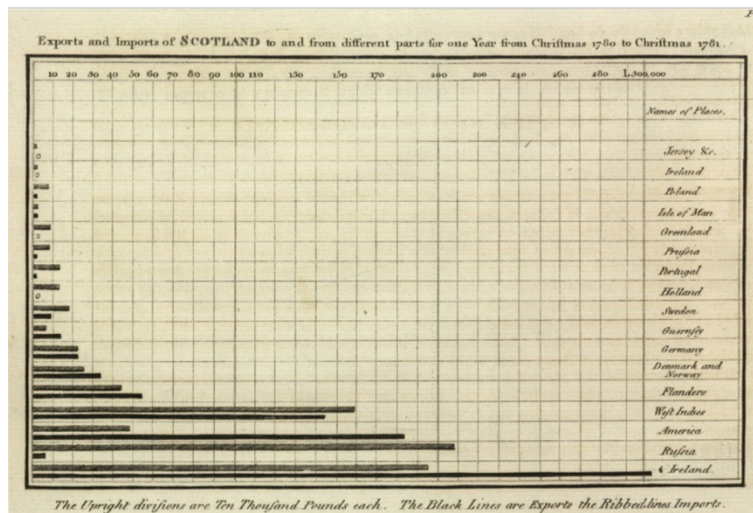
*Whatever relates to extent and quantity may be represented by geometrical figures. Statistical projections which speak to the senses without fatiguing the mind, possess the advantage of fixing the attention on a great number of important facts.*

*—Alexander von Humboldt, 1811*

# A very brief history of data visualization

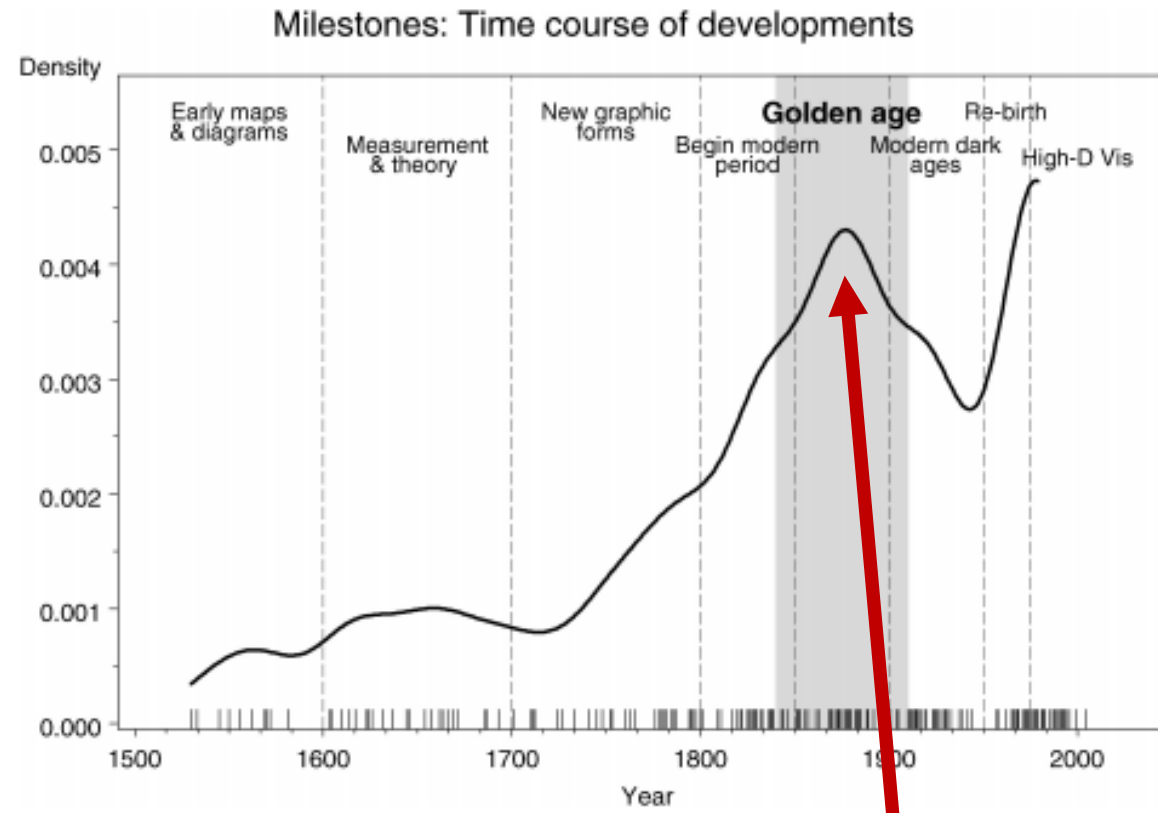
The age of modern statistical graphs began around the beginning of the 19<sup>th</sup> century

[William Playfair](#) (1759-1823) is credited with inventing the line graph, bar chart and pie chart



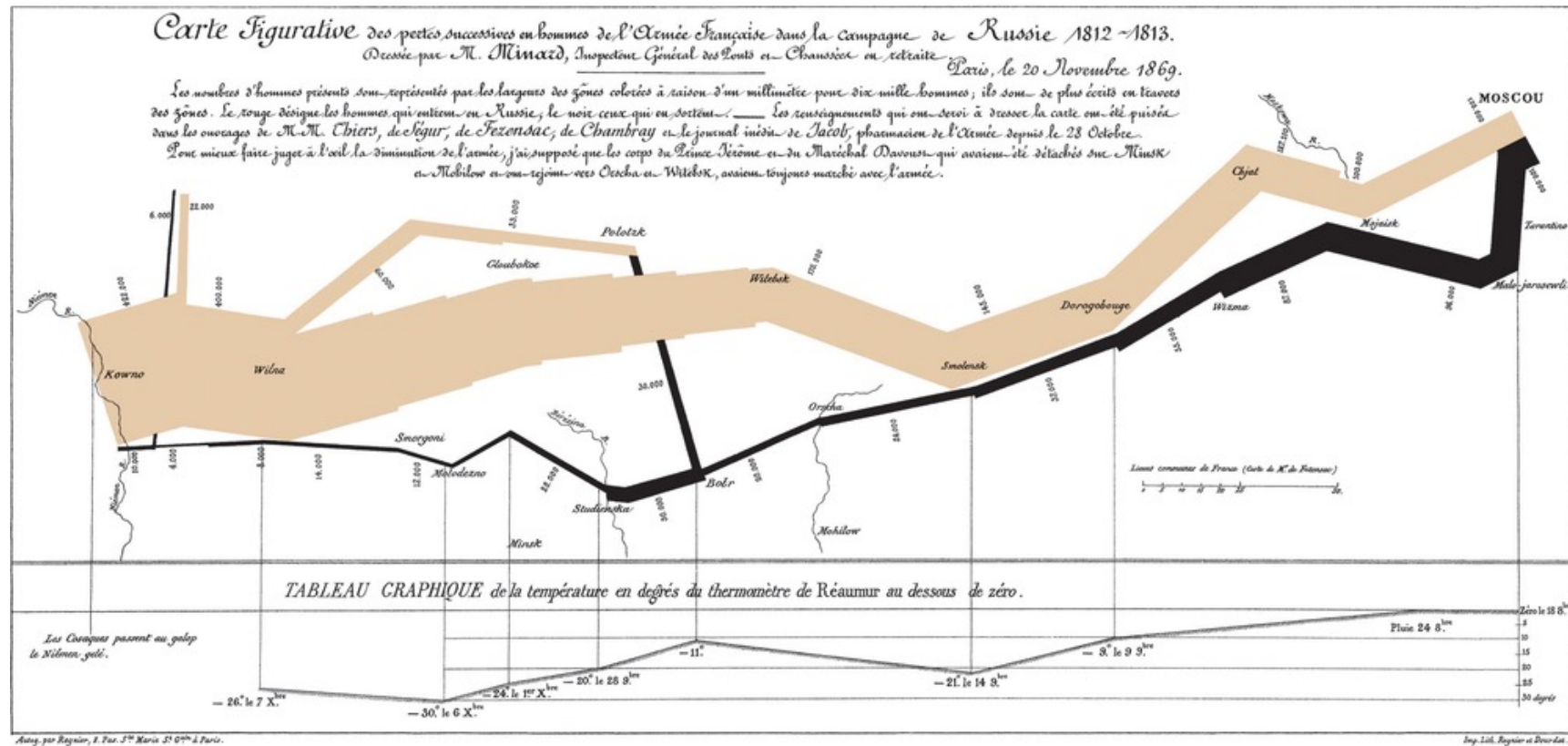
# A very brief history of data visualization

According to Friendly, statistical graphics researched its golden age between 1850-1900



# A very brief history of data visualization

## Joseph Minard (1781-1870)

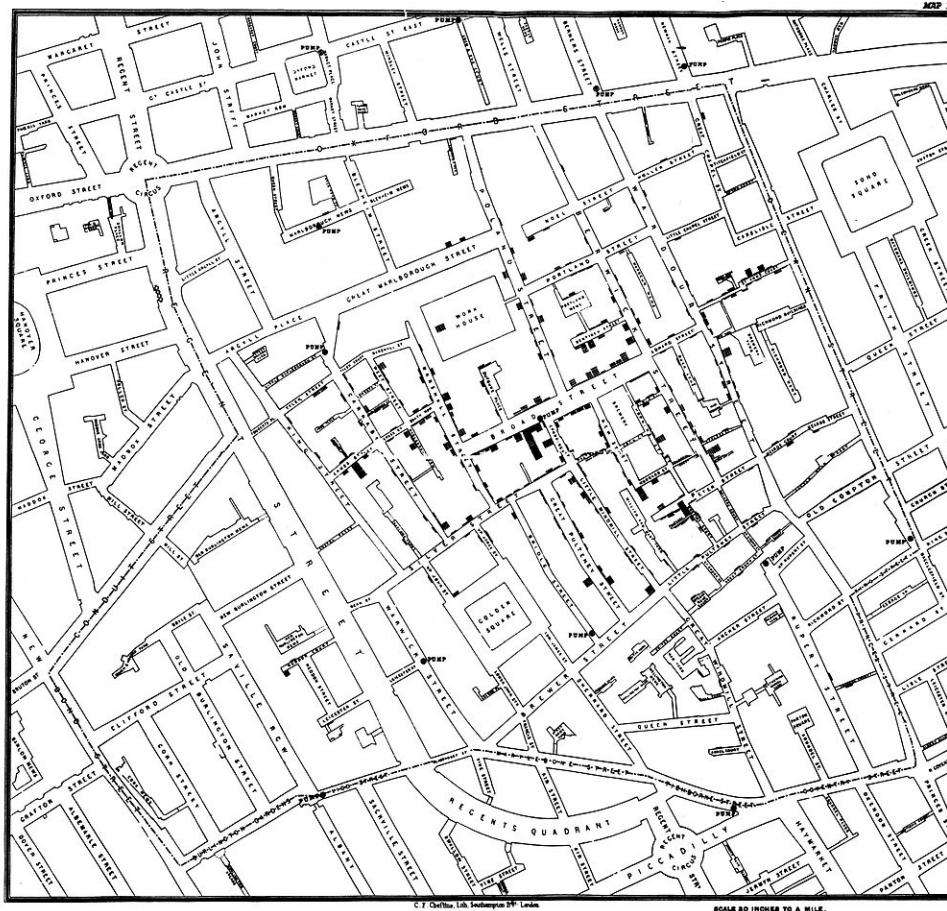


Map of Napoleon's march on Russia



# A very brief history of data visualization

John Snow (1813-1858)



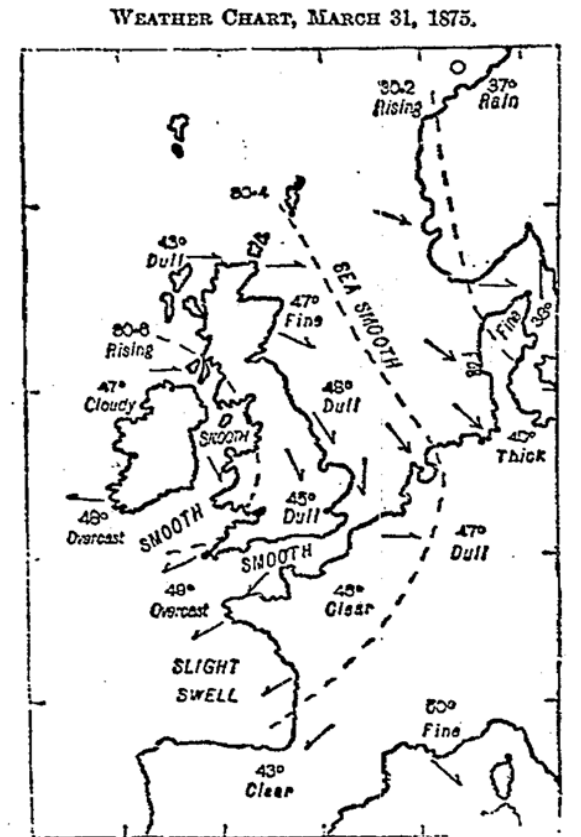
Clusters of cholera  
cases in London  
epidemic of 1854

### Diagram of the causes of mortality in the army in the east



# A very brief history of data visualization

Francis Galton (1822-1911)

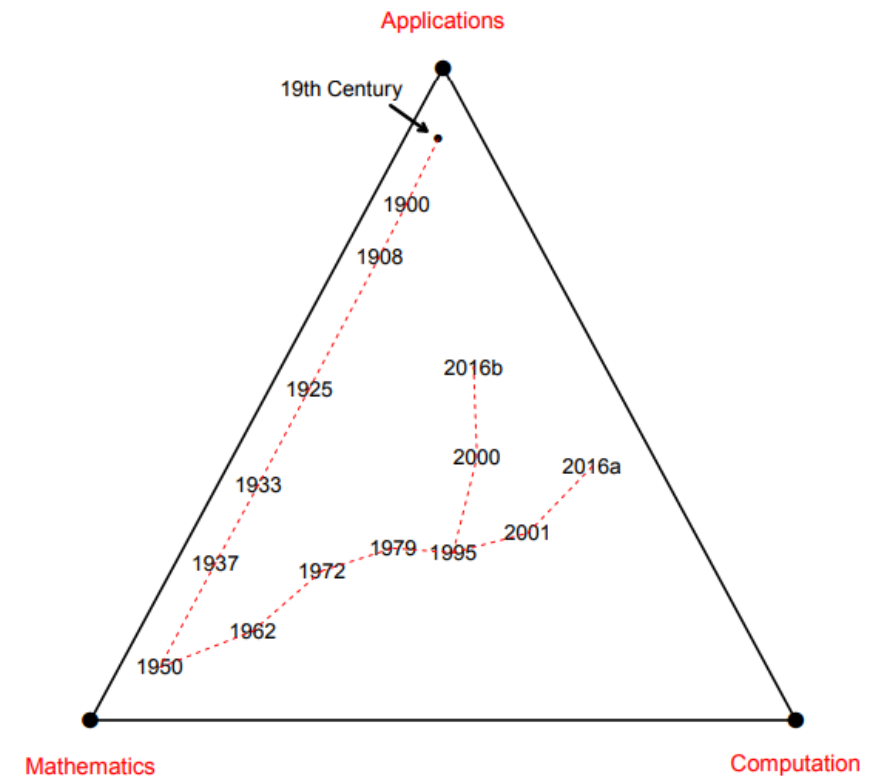
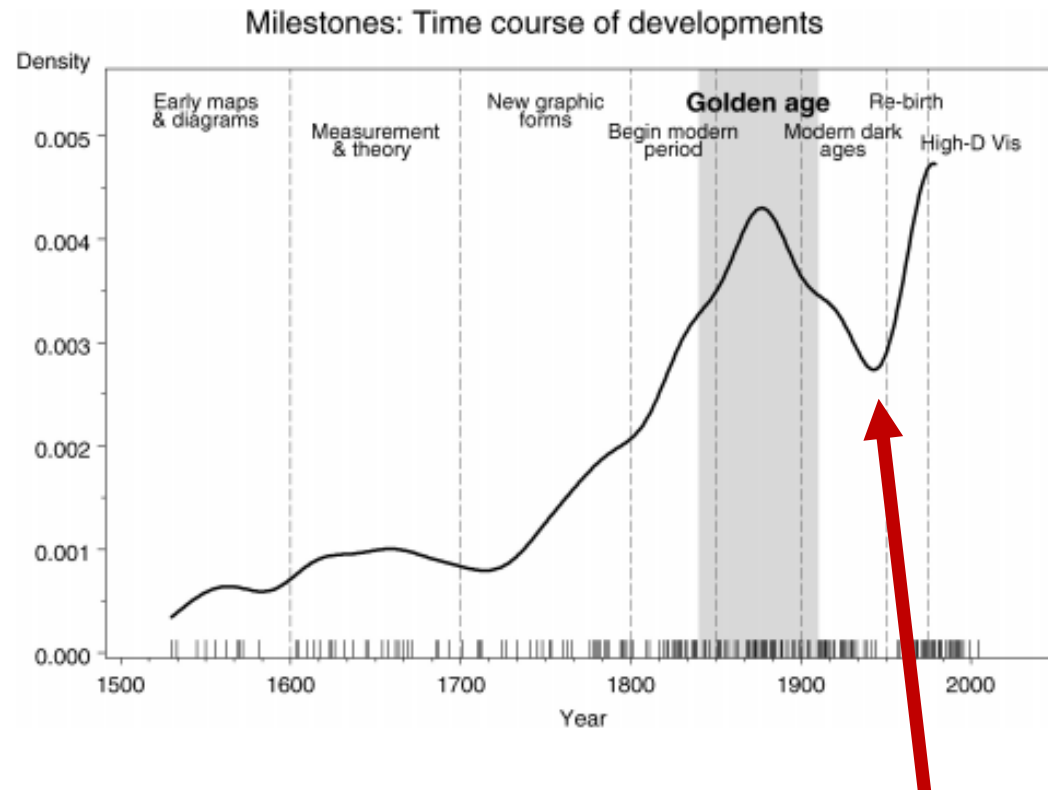


The dotted lines indicate the gradations of barometric pressure. The variations of the temperature are marked by figures, the state of the sea and sky by descriptive words, and the direction of the wind by arrows—barbed and feathered according to its force. ○ denotes calm.

First weather map published in a newspaper (1875)

# A very brief history of data visualization

“Graphical dark ages” around 1950



Computer Age Statistical Inference, Efron and Hastie

# A very brief history of data visualization

Currently undergoing a “Graphical re-birth”

Box plot

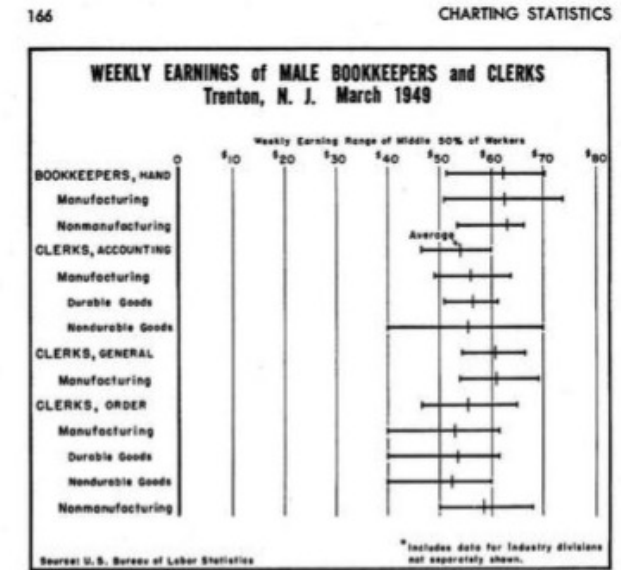
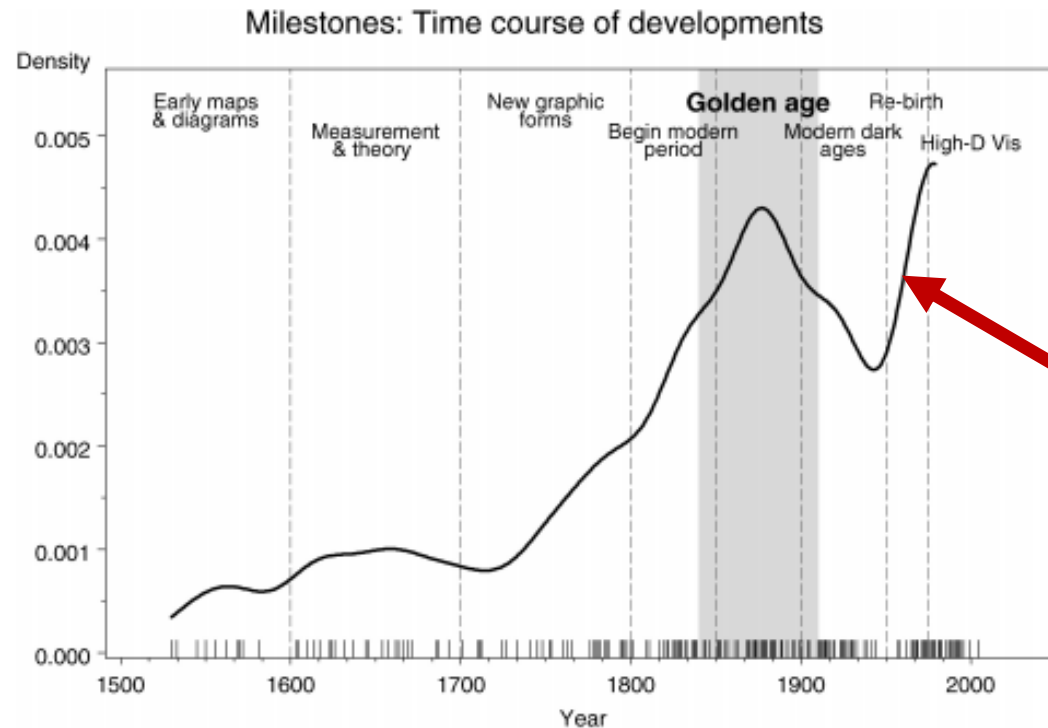


Fig. 6-23. The range bar and symbol.

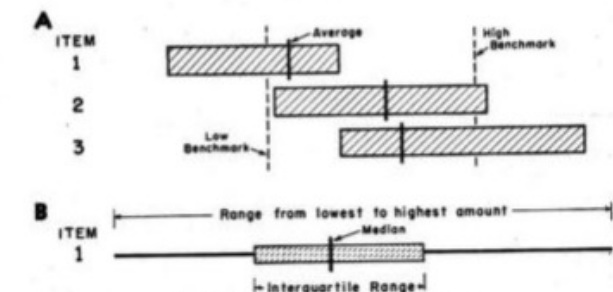
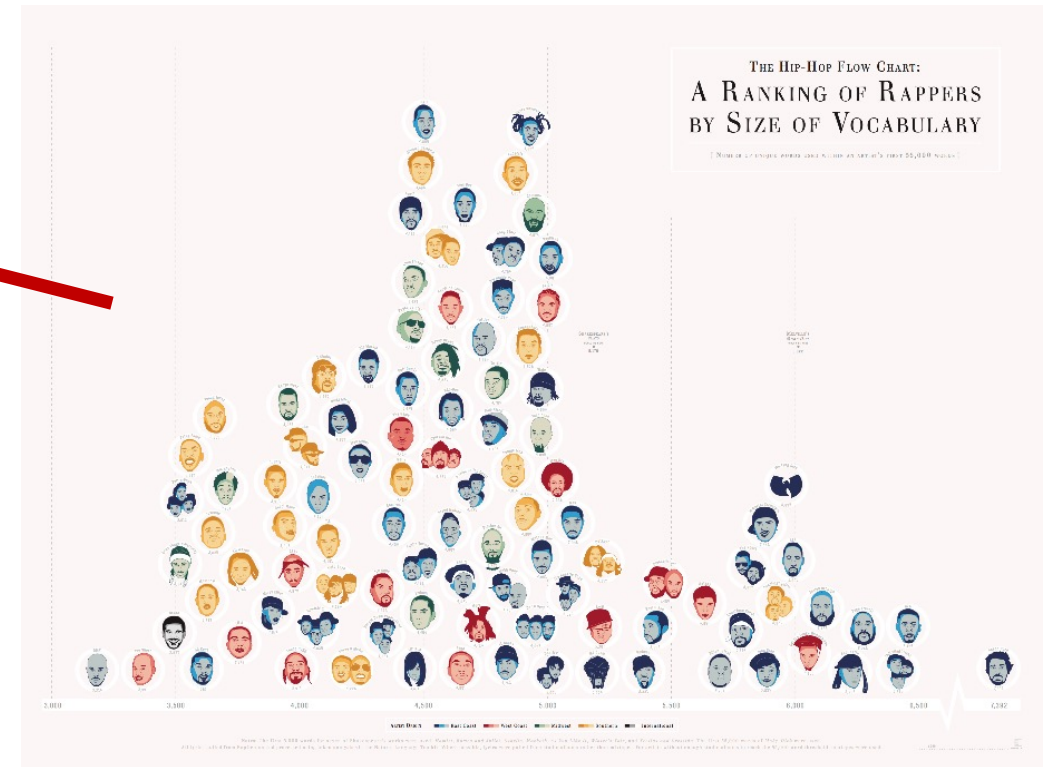
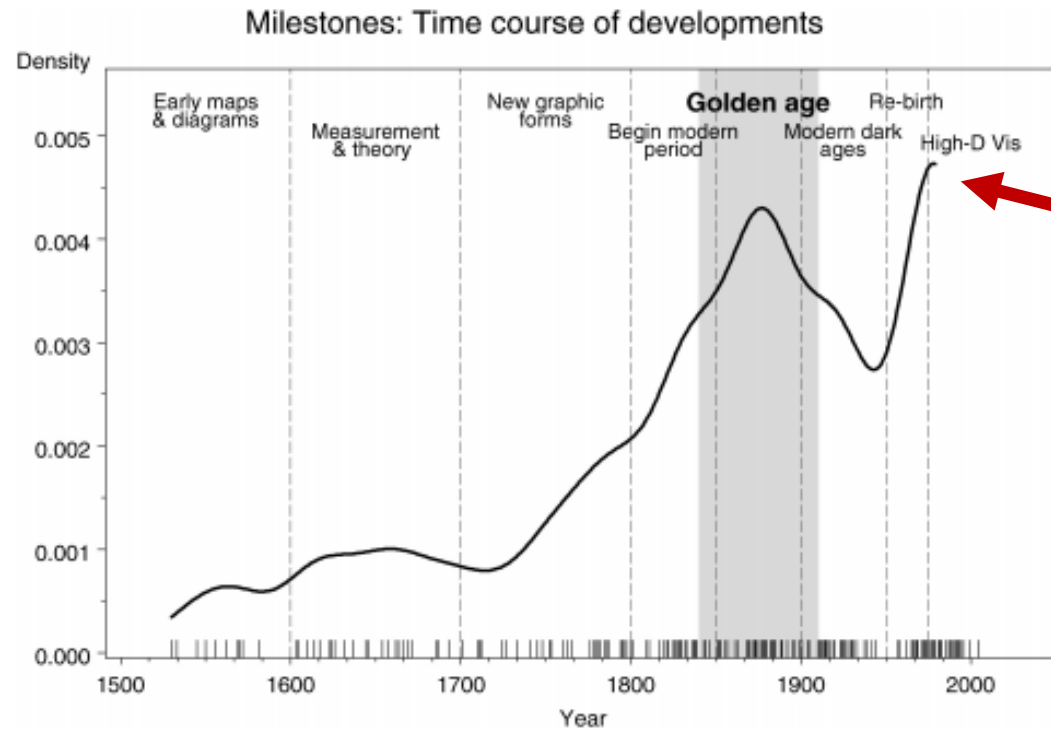


Fig. 6-24. Various uses of the range bar.

[Spear 1952](#), [Tukey 1970](#)

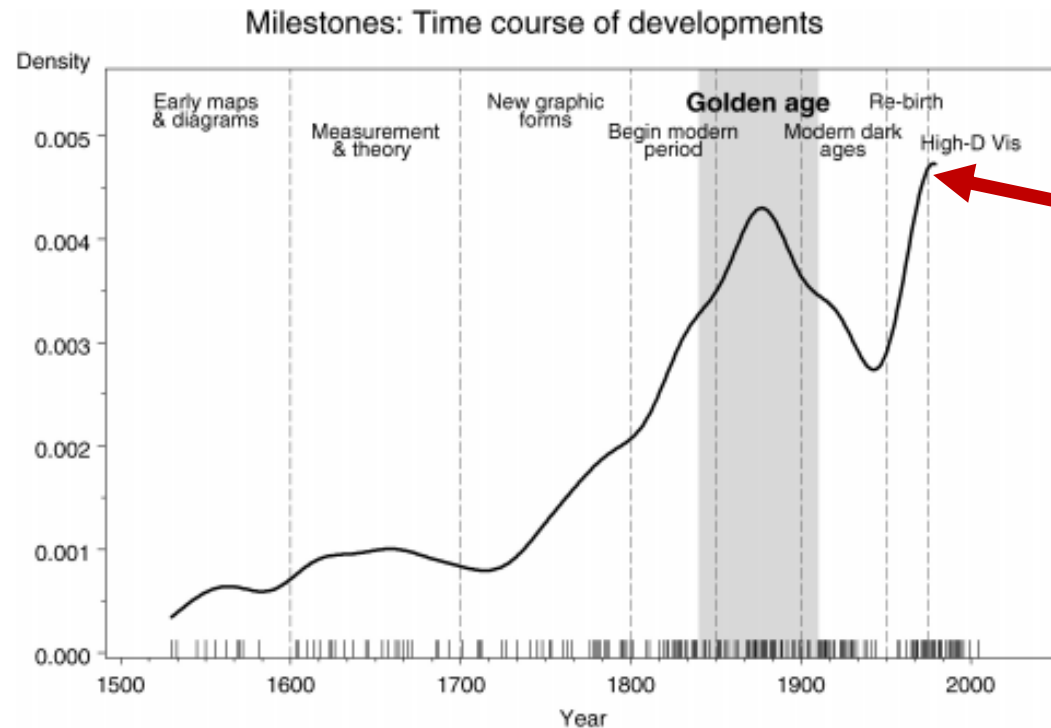
# A very brief history of data visualization

Currently undergoing a “Graphical re-birth”



# A very brief history of data visualization

Currently undergoing a “Graphical re-birth”

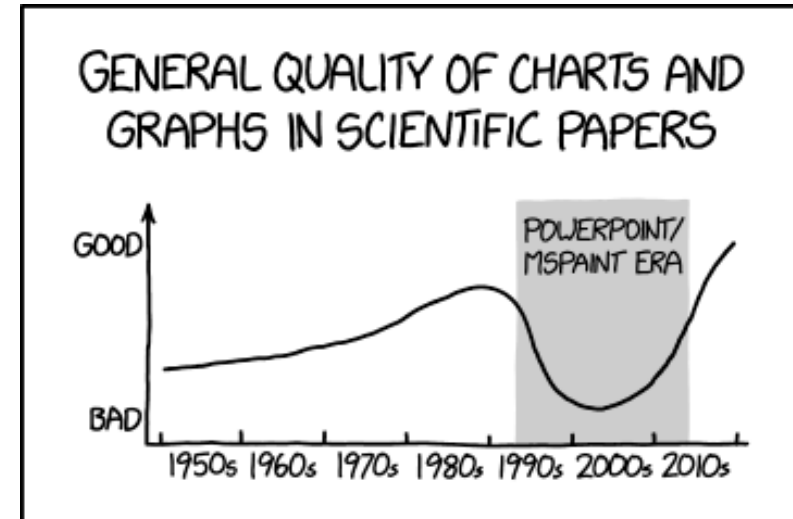
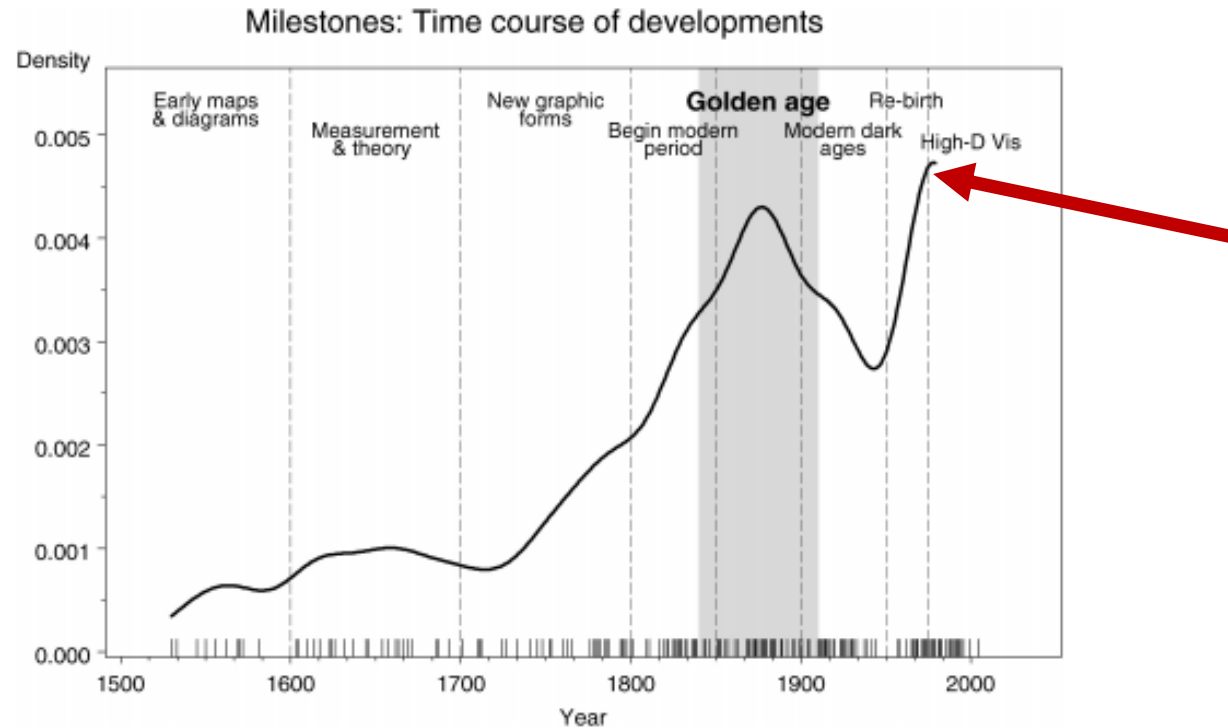


Hans Rosling's gapminder

- [Simple version](#)
- [TV special effects](#)
- [Ted Talk](#)

# A very brief history of data visualization

Currently undergoing a “Graphical re-birth”

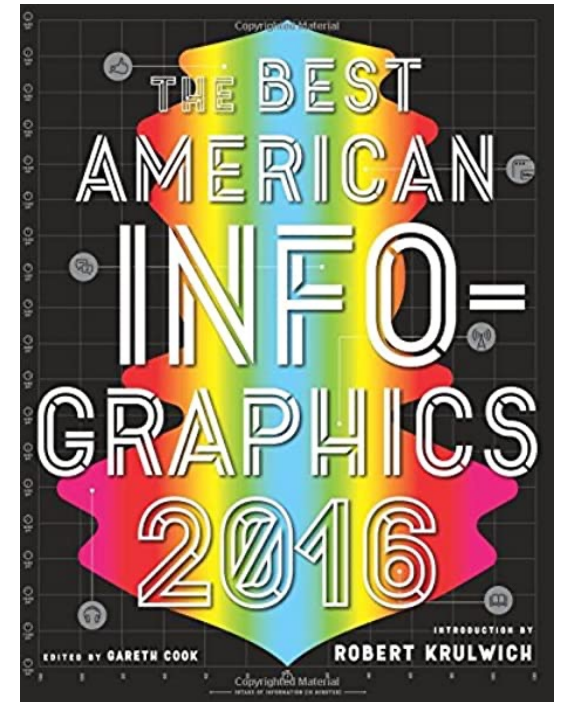


# Coming up on homework 5: find an interesting data visualization...

Homework 5 : Find an interesting data visualization

- <https://www.reddit.com/r/dataisbeautiful/>
- <https://flowingdata.com/>

We will do a little show and tell in class



# Visualizing data with matplotlib

[Matplotlib](#) is a comprehensive library for creating static, animated, and interactive visualizations in Python.

- Matplotlib makes easy things easy and hard things possible.

Note: there are two different "interfaces" to matplotlib, that use slightly different syntax

- An explicit "Axes" interface
- An implicit "pyplot" interface
  - Just be aware if you are reading Stack overflow articles

We will use the pyplot interface (for now)

```
import matplotlib.pyplot as plt
```



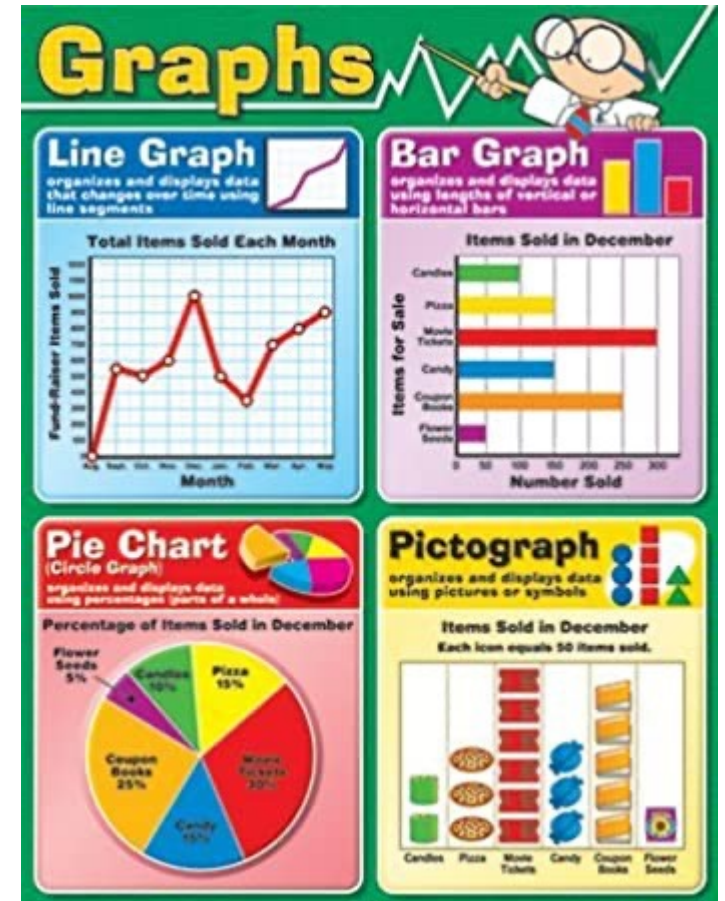


# Types of plots

There are many different ways to plot data, many of which I am sure you have seen before

The type of plot you choose will depend on the type of data you have and what you want to emphasize

- i.e., There are different types of plots of categorical data, a single quantitative variable, two quantitative variables, etc.
  - This will become even more apparent when we look at the seaborn package

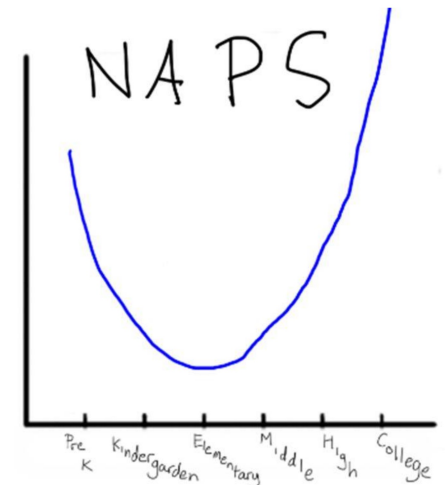


# Line graph

Line graph (line chart, or curve chart) displays information as a series of data points called "markers" connected by straight line segments.

We can create line graphs in matplotlib using:

- `plt.plot(x)` # create a line graph with successive integers as the x-values
- `plt.plot(x, y)` # plots y as a function of x
- `plt.plot(x, y, '-o')` # creates lines with circle markers



Let's explore this in Jupyter!

# Visualizing quantitative data: histograms

Salaries of basketball players (in millions of dollars):

- 2.53, 18.6, 9.4, 21.7, ...

To create a histogram we create a set of intervals

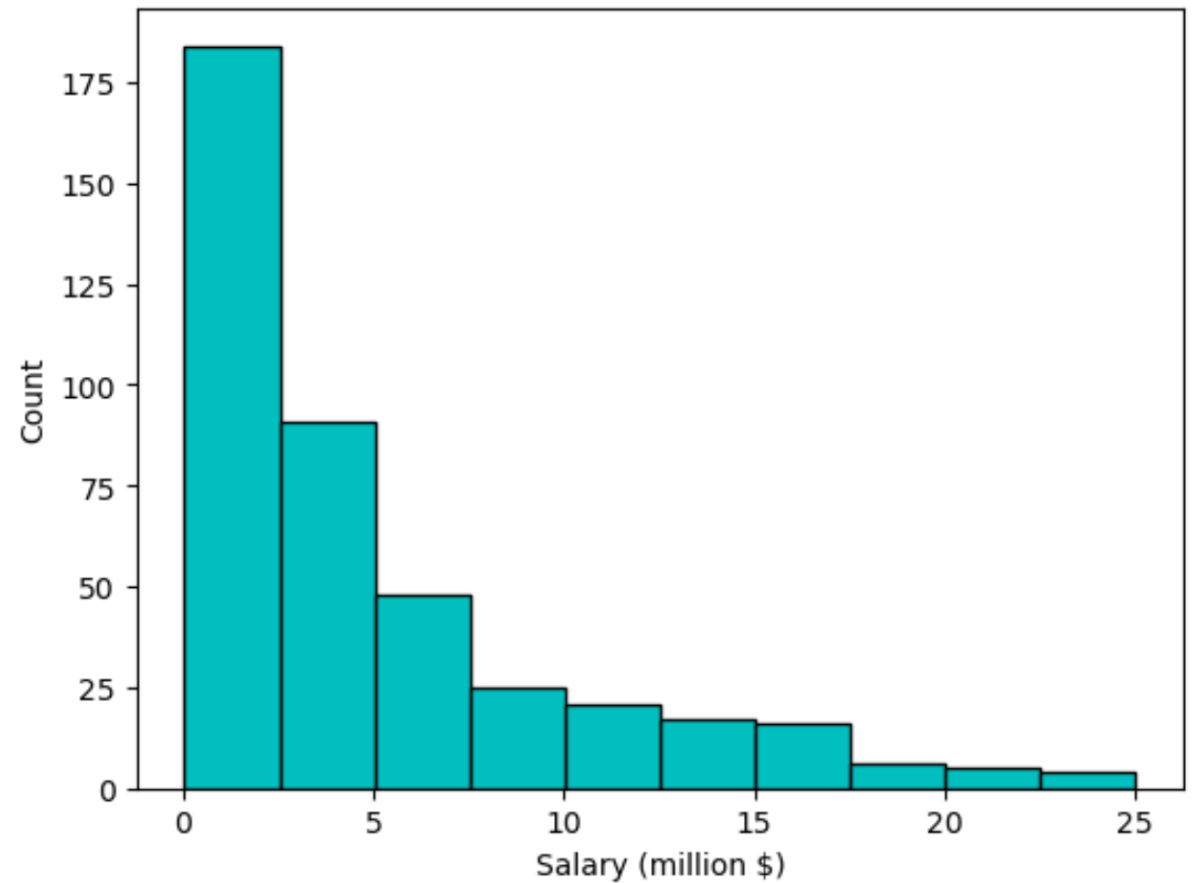
- 0-2.5, 2.5-5, 5-7.5, ... 20-22.5, 22.5-25.0

We count the number of points that fall in each interval

We create a bar chart where the height of the bars is the counts in each bin

# Histograms – countries life expectancy in 2007

Life Expectancy	Frequency Count
(0 – 2.5]	184
(2.5 – 5]	91
(5 – 7.5]	48
(7.5 – 10]	25
(10 – 12.5]	21
(12.5 – 15]	17
(15 – 17.5]	16
(17.5 – 20]	6
(20 – 22.5]	5
(22.5 – 25]	4



`plt.hist(data)`

To be continued...

