



# Data science

Pandas Explore

2025-2026



# Pandas explore

---

- Adding 1 row to DataFrame
- Combining datasets
- Reformat data
- Overview in graphs
- Boxplots, skewing and distributions



# Data exploration

- We've done data import and data selection in pandas in the previous chapter.
- Those were the tools, in this chapter we're talking about how to use these tools when confronted with a new dataset.
- And just a warning: in the course “Data visualisation” you will learn which graphs are appropriate in which case
  - E.g. You can only use a line graph when plotting continuous variables
- We'll try not to break those rules, but we may stray from the correct path



# Adding 1 row to DataFrame

- Using “.concat”

```
#Example DataFrame  
data = {"Name": ["Alice", "Bob", "Charlie", "David", "Eve"],  
        "Age": [25, 30, 35, 40, 45]}  
df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])  
print(df)
```

	Name	Age
a	Alice	25
b	Bob	30
c	Charlie	35
d	David	40
e	Eve	45

```
# new row to add  
new_row = {"Name": "Frank", "Age": 50}  
# adding the new row  
df = pd.concat([df, pd.DataFrame([new_row], index=['f'])])  
print("\nDataFrame after adding new row:")  
print(df)
```

	Name	Age
a	Alice	25
b	Bob	30
c	Charlie	35
d	David	40
e	Eve	45
f	Frank	50



# Adding 1 row to DataFrame

- Using .loc
  - new label → new entry
  - existing label → overwrite entry

```
#alternative method using loc
df.loc['g'] = {"Name": "Grace", "Age": 55}
print("\nDataFrame after adding new row using loc:")
print(df)
```

```
# alternative method using loc with existing index
df.loc['f'] = {"Name": "Hank", "Age": 60}
print("\nDataFrame after adding new row using existing index loc:")
print(df)
```

	Name	Age
a	Alice	25
b	Bob	30
c	Charlie	35
d	David	40
e	Eve	45
f	Frank	50
g	Grace	55

	Name	Age
a	Alice	25
b	Bob	30
c	Charlie	35
d	David	40
e	Eve	45
f	Hank	60
g	Grace	55



# Adding 1 row to DataFrame

- Using `iloc` throws error → only works with existing rows

```
#adding a new row using iloc will raise an IndexError  
df.iloc[7] = {"Name": "Ivy", "Age": 65}
```

`IndexError`: `iloc` cannot enlarge its target object

```
#using an existing index will overwrite the row  
df.iloc[3] = {"Name": "Ivy", "Age": 65}
```

	Name	Age
a	Alice	25
b	Bob	30
c	Charlie	35
d	Ivy	65
e	Eve	45
f	Hank	60
g	Grace	55



# Combining multiple datasets

- Real-world data often comes in multiple pieces. You need to know how to grab those pieces and combine them into one dataset and make it ready for analysis.
- What the pieces are (pandas Series or DataFrames) may differ
- We start with **2 Series:** to create **1 DataFrame:**

```
city_rank  
Kabul      36  
Tirana     138  
Algiers    34  
dtype: int64
```

```
city_2022_population  
Kabul      41128771  
Tirana     2842321  
dtype: int64
```

```
city_data  
Rank 2022 population  
Algiers      34        NaN  
Kabul       36  41128771.0  
Tirana      138   2842321.0
```

```
city_data = pd.DataFrame({"Rank": city_rank,"2022 population": city_2022_population})
```



# Combining multiple datasets

- Next up is combining 2 DataFrames with identical columns
- In this example we concatenate 2 DataFrames into a new one

`city_data`

	Rank	2022 population
Algiers	34	NaN
Kabul	36	41128771.0
Tirana	138	2842321.0

`further_city_data`

	Rank	2022 population
Pago Pago	213	44273
Andorra la Vella	203	79824

```
all_city_data = pd.concat([city_data, further_city_data], sort=False)
```

`all_city_data`

	Rank	2022 population
Algiers	34	NaN
Kabul	36	41128771.0
Tirana	138	2842321.0
Pago Pago	213	44273.0
Andorra la Vella	203	79824.0



# Combining multiple datasets

- What happens when they share a key, but have different columns?

city\_countries

	country	capital	
Amsterdam	Holland	1	
Tokyo	Japan	1	
Tirana	Albanië	1	
Rotterdam	Holland	0	
Toronto	Canada	0	
Barcelona	Spain	0	

all\_city\_data

	Rank	2022 population	
Algiers	34	Nan	
Kabul	36	41128771.0	
Tirana	138	2842321.0	
Pago Pago	213	44273.0	
Andorra la Vella	203	79824.0	

```
cities = pd.concat([all_city_data, city_countries], axis=1, sort=False)
```

cities

	Rank	2022 population	country	capital
Algiers	34.0	Nan	Nan	Nan
Kabul	36.0	41128771.0	Nan	Nan
Tirana	138.0	2842321.0	Albanië	1.0
Pago Pago	213.0	44273.0	Nan	Nan
Andorra la Vella	203.0	79824.0	Nan	Nan
Amsterdam	Nan	Nan	Holland	1.0
Tokyo	Nan	Nan	Japan	1.0
Rotterdam	Nan	Nan	Holland	0.0
Toronto	Nan	Nan	Canada	0.0
Barcelona	Nan	Nan	Spain	0.0

All rows are kept.  
NaN appears when no data available



# Combining multiple datasets

- We can prevent the Na's by using datasets with combinable data (~key)

capital\_countries

	Country	Capital
Kabul	Afghanistan	Kabul
Tirana	Albania	Tirana
Algiers	Algeria	Algiers
Pago Pago	American Samoa	Pago Pago
Andorra la Vella	Andorra	Andorra la Vella

all\_city\_data

	Rank	2022 population
Algiers	34	NaN
Kabul	36	41128771.0
Tirana	138	2842321.0
Pago Pago	213	44273.0
Andorra la Vella	203	79824.0

```
capitals = pd.concat([all_city_data, capital_countries], axis=1, sort=False)
```

capitals

	Rank	2022 population	Country	Capital
Algiers	34	NaN	Algeria	Algiers
Kabul	36	41128771.0	Afghanistan	Kabul
Tirana	138	2842321.0	Albania	Tirana
Pago Pago	213	44273.0	American Samoa	Pago Pago
Andorra la Vella	203	79824.0	Andorra	Andorra la Vella



# Merge

- Merging means that you combine two datasets that don't share a key column
- You choose a column in one DataFrame to link with the key of the other DataFrame

**cities**

	Rank	2022 population	country	capital
Algiers	34.0	NaN	NaN	NaN
Kabul	36.0	41128771.0	NaN	NaN
Tirana	138.0	2842321.0	NaN	NaN
Pago Pago	213.0	44273.0	NaN	NaN
Andorra la Vella	203.0	79824.0	NaN	NaN
Amsterdam	NaN	NaN	Holland	1.0
Tokyo	NaN	NaN	Japan	1.0
Rotterdam	NaN	NaN	Holland	0.0
Toronto	NaN	NaN	Canada	0.0
Barcelona	NaN	NaN	Spain	0.0

**countries**

	population_millions	continent
Holland	17	Europe
Japan	127	Asia
Canada	37	North America
Belgium	11	Europe



# Merge syntax

```
pd.merge(countries, cities, right_on="country", left_index=True)
pd.merge(cities, countries, left_on="country", right_index=True)
```

- In the example:
  - DataFrame countries contains the index
  - DataFrame cities contains a column ‘country’
- Begin by selecting both DataFrames. The order is important!
- With “left\_index” or “right\_index” you decide which of the 2 DataFrames contains the index
  - pd.merge (**countries**, cities) → left\_index = True
  - pd.merge (cities, **countries**) → right\_index = True
- With “left\_on” or “right\_on” you decide with which column in the other DataFrame (right of left) the index column must merge
  - pd.merge (countries, **cities**, left\_index = True) → right\_on = country'
  - pd.merge (**cities**, countries, right\_index = True) → left\_on = ‘country’



# Merge example

countries

	population_millions	continent
Holland	17	Europe
Japan	127	Asia
Canada	37	North America
Belgium	11	Europe

```
pd.merge(countries,cities, left_index=True, right_on='country')
```

cities

	Rank	2022 population	country	capital
Algiers	34.0	Nan	Nan	Nan
Kabul	36.0	41128771.0	Nan	Nan
Tirana	138.0	2842321.0	Nan	Nan
Pago Pago	213.0	44273.0	Nan	Nan
Andorra la Vella	203.0	79824.0	Nan	Nan
Amsterdam	Nan	Nan	Holland	1.0
Tokyo	Nan	Nan	Japan	1.0
Rotterdam	Nan	Nan	Holland	0.0
Toronto	Nan	Nan	Canada	0.0
Barcelona	Nan	Nan	Spain	0.0

	Rank	2022 population	country	capital	population_millions	continent
Amsterdam	Nan	Nan	Holland	1.0	17	Europe
Rotterdam	Nan	Nan	Holland	0.0	17	Europe
Tokyo	Nan	Nan	Japan	1.0	127	Asia
Toronto	Nan	Nan	Canada	0.0	37	North America

- Notice how “Belgium” is not retained as it doesn’t have a corresponding row in cities
- Same for Algiers, Kabul, ... they don’t exist in countries



# Merge as inner join

- Do watch out: we merged two DataFrames with 4 and 10 rows, and got a 4 row DataFrame as result!
- This is because we inner joined.  
You'll lose rows that don't have a match in the other DataFrame's key column.
- Also note: all columns were kept, even those that now only contain NaN-values.



# Merge left and right

- By default, merge() always performs an inner join.
- However, by using the attribute ‘how’ you can indicate the type of join you want to perform

```
pd.merge(cities, countries, how='left', left_on="country", right_index=True)
```

	Rank	2022 population	country	capital	population_millions	continent
Algiers	34.0	NaN	NaN	NaN	NaN	NaN
Kabul	36.0	41128771.0	NaN	NaN	NaN	NaN
Tirana	138.0	2842321.0	NaN	NaN	NaN	NaN
Pago Pago	213.0	44273.0	NaN	NaN	NaN	NaN
Andorra la Vella	203.0	79824.0	NaN	NaN	NaN	NaN
Amsterdam	NaN	NaN	Holland	1.0	17.0	Europe
Tokyo	NaN	NaN	Japan	1.0	127.0	Asia
Rotterdam	NaN	NaN	Holland	0.0	17.0	Europe
Toronto	NaN	NaN	Canada	0.0	37.0	North America



# Reformat data

---

- Data isn't always nice and rectangular
  - Or it is, but it's the wrong type of rectangle
  - Or in two rectangles (aka files) that need to be combined
- You can reformat your data into the form you need
- Another example of reshaping is a pivot table
- Pivoting allows you to summarize your data based on certain properties
  - Helps a lot to get a quick overview of what is going on



# DataFrame reshaping

- `melt()`:
  - make wide data long
- `pivot()`:
  - make long data wide
- `pivot_table()`:
  - same as `.pivot()`  
but can handle multiple indexes
- `crosstab()`

	wide		
id	x	y	z
1	a	c	e
2	b	d	f



# Melt

- Used to collapse a DataFrame so that every column becomes a new row
- There are some `id_vars`, they will be copied
- The names of the columns are kept as values

Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

`df3.melt(id_vars=['first', 'last'])`

→

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150



# Melt example

```
1 df = pd.DataFrame({"Name": ["Tom", "Glen", "Sara", "Kian"],  
2 | | | | "2018": [3, 4, 5, 3],  
3 | | | | "2019": [4, 3, 2, 1],  
4 | | | | "2020": [2, 4, 4, 3]})  
5 df
```

	Name	2018	2019	2020
0	Tom	3	4	2
1	Glen	4	3	4
2	Sara	5	2	4
3	Kian	3	1	3

```
1 df_melt = df.melt(id_vars="Name", var_name="Year", value_name="Courses")  
2 df_melt
```

	Name	Year	Courses
0	Tom	2018	3
1	Glen	2018	4
2	Sara	2018	5
3	Kian	2018	3
4	Tom	2019	4
...	...	...	...
7	Kian	2019	1
8	Tom	2020	2
9	Glen	2020	4
10	Sara	2020	4
11	Kian	2020	3

12 rows × 3 columns



# Melt, other example

- Suppose you were given the following DataFrame

	Student	Math	English	Science
0	Alice	90	85	95
1	Bob	80	75	85
2	Charlie	70	65	75

- Can you (easily) calculate the average score for every student?
    - No, because they are in different columns
    - `df["avg"] = (df["Math"] + df["English"] + df["Science"]) / 3`
- Not maintainable code!



# Melt, other example

- Solution: melt!

```
df_melted = pd.melt(df, id_vars=['Student'], value_vars=['Math', 'English', 'Science'], var_name='Subject', value_name='Score')
```

	Student	Math	English	Science
0	Alice	90	85	95
1	Bob	80	75	85
2	Charlie	70	65	75



	Student	Subject	Score
0	Alice	Math	90
1	Bob	Math	80
2	Charlie	Math	70
3	Alice	English	85
4	Bob	English	75
5	Charlie	English	65
6	Alice	Science	95
7	Bob	Science	85
8	Charlie	Science	75

- The average per student can now be calculated using group by

```
df_mean = df_melted.groupby('Student')['Score'].mean().reset_index()
```



# Pivot

- Pivot is basically the opposite of melt
- It combines multiple rows and uses one of the values as column name and the other as value
- Watch out, other columns may be dropped in the process
  - Like “zoo” in the example

Pivot

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

`df.pivot(index='foo',  
 columns='bar',  
 values='baz')`

→

bar	A	B	C
foo			
one	1	2	3
two	4	5	6



# Pivot example

- The simple example would be to un-melt the DataFrame we made before

```
1 df_melt  
✓ 0.4s
```

	Name	Year	Courses
0	Tom	2018	3
1	Glen	2018	4
2	Sara	2018	5
3	Kian	2018	3
...	...	...	...
8	Tom	2020	2
9	Glen	2020	4
10	Sara	2020	4
11	Kian	2020	3

12 rows × 3 columns

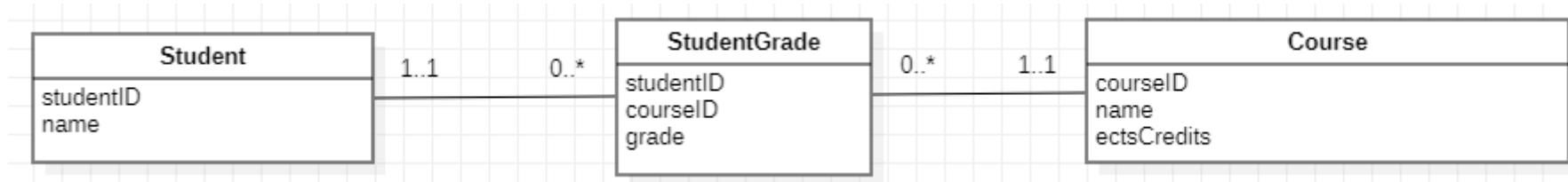
```
1 df_pivot = df_melt.pivot(index="Name",columns="Year",values="Courses")  
2 df_pivot  
✓ 0.5s
```

Year	2018	2019	2020
Name			
Glen	4	3	4
Kian	3	1	3
Sara	5	2	4
Tom	3	4	2



# Pivot usage

- A lot of numerical data is stored in SQL-databases
- SQL-databases are normalized, so that would look like this:



- It's impossible to write a SQL-query where the rows of the table “Course” become the column names
- But when you want to use the grades of students to do machine learning, you need all the grades on one line
  - Pivot!



# Pivot and summarize

- Up until now we only used pivot to copy values
  - The example: every combination of “Name” and “Year” was unique
- But pivot can also create summaries
  - What is the average number of courses per student?
  - What is the total number of courses per student?
  - What is the standard deviation of the number of courses students took per year?
- That is when we start using a pivot table
  - You have this in Excel as well

	Name	Year	Courses
0	Tom	2018	3
1	Glen	2018	4
2	Sara	2018	5
3	Kian	2018	3

# Pivot table

A Pivot table:

- is used to summarize, sort, reorganize, group, count, total or average data stored in a table.
- allows us to transform columns into rows and rows into columns.
- allows grouping by any field (column) and performing advanced calculations on them.

## PIVOT TABLES CHEATSHEET

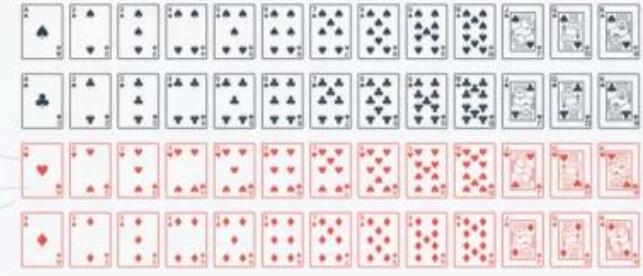
Let's start with the standard deck of 52 card. Each card has some properties (attributes):

Value (e.g. A)

Symbol (e.g. ♠)

Color (e.g. red)

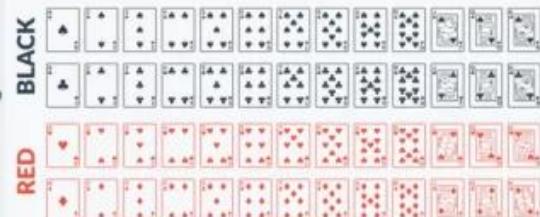
### The Standard Deck of 52 Cards



### GROUPING THE DECK by the card color

We split the deck into two. We could also split the cards by any other attribute - value or symbol.

How many cards are there in each color group/category?



### OUR FIRST PIVOT TABLE

We have just counted the black and red cards

BLACK 26    RED 26

Labels in a row or column are just about our preference; what tells our story better.

BLACK 26  
RED 26

BLACK 26    RED 26

ROTATE

### FINAL TOUCH

We add headers so that the readers know what they are looking at.

Color	Count of Card
BLACK	26
RED	26

### HOW DO WE...

...put the cards in the computer?

Computers need tabular data, so we describe each card as a row in a table

Value	Symbol	Color
A	♥	RED
2	♥	RED
3	♥	RED
...	...	...



# Pivot table: pivot()

- What we start with: Minimum and maximum temperatures for some cities
- The cities we have:
  - ['Brussels', 'Amsterdam', 'London', 'Berlin', 'Madrid', 'Sydney']

	City	Month	Minimum_temperature	Maximum_temperature
0	Brussels	January	1.2	6.3
1	Brussels	February	1.3	7.2
2	Brussels	March	3.1	10.8
3	Brussels	April	5.2	14.8
4	Brussels	May	8.4	18.4



# Pivot table: pivot()

- “Unfold” the table by using pivot()
  - Cities vs months or the other way around

```
df_pivot = df.pivot(index="City",columns="Month",values="Minimum_temperature")  
df_pivot = df.pivot(index="Month",columns="City",values="Minimum_temperature")
```

Month	April	August	December	February	January	July	June	March
City								
Amsterdam	4.0	13.0	2.0	0.5	1.0	13.0	11.0	2.0
Berlin	5.0	14.0	-0.5	-1.0	0.5	14.0	12.0	2.0
Brussels	5.2	13.3	1.8	1.3	1.2	13.4	11.3	3.1
London	5.0	13.0	3.0	1.0	2.0	13.0	11.0	3.0
Madrid	7.4	15.6	2.6	2.5	1.9	18.6	15.8	5.1
Sydney	14.9	9.5	17.5	18.7	18.5	16.4	19.3	17.6

City	Amsterdam	Berlin	Brussels	London	Madrid	Sydney
Month						
April	4.0	5.0	5.2	5.0	7.4	14.9
August	13.0	14.0	13.3	13.0	18.6	9.5
December	2.0	-0.5	1.8	3.0	2.6	17.5
February	0.5	-1.0	1.3	2.0	2.5	18.7
January	1.0	0.5	1.2	2.0	1.9	18.5
July	13.0	14.0	13.4	13.0	18.6	8.4
June	11.0	12.0	11.3	11.0	15.8	9.3
March	2.0	-0.5	3.1	3.0	5.1	17.6

The order of the months is wrong!  
We lose the maximum temperature!



# Pivot table - revisited

- “Unfold” the table using pivot()
  - Cities vs months or the other way around
  - Months appear chronologically
  - Average temperature in cells instead of min or max temperatures

Month	January	February	March	April	May	June	July	August
City								
Amsterdam	3.00	3.25	5.50	8.00	12.5	15.50	17.50	17.50
Berlin	0.50	0.50	4.50	9.00	14.0	17.00	19.00	19.00
Brussels	3.75	4.25	6.95	10.00	13.4	16.30	18.45	18.30
London	5.00	5.00	7.00	9.50	13.0	16.00	18.00	18.00
Madrid	6.35	7.65	10.75	13.20	17.4	22.80	26.00	25.75
Sydney	22.25	22.30	21.20	18.55	15.4	12.95	12.20	13.50

City	Amsterdam	Berlin	Brussels	London	Madrid	Sydney
Month						
January	3.00	0.50	3.75	5.0	6.35	22.25
February	3.25	0.50	4.25	5.0	7.65	22.30
March	5.50	4.50	6.95	7.0	10.75	21.20
April	8.00	9.00	10.00	9.5	13.20	18.55
May	12.50	14.00	13.40	13.0	17.40	15.40
June	15.50	17.00	16.30	16.0	22.80	12.95
July	17.50	19.00	18.45	18.0	26.00	12.20
August	17.50	19.00	18.30	18.0	25.75	13.50

What did we do to arrange the months chronologically?



# Pivot table with aggregations: pivot\_table()

- What is the average (mean) temperature per city or per month?

```
table = pd.pivot_table(df, values='Average_temperature', index=['City'], aggfunc="mean")
table = pd.pivot_table(df, values='Average_temperature', index=['Month'], aggfunc="mean")
```

Average_temperature	
City	
Amsterdam	9.854167
Berlin	9.395833
Brussels	10.787500
London	10.958333
Madrid	15.345833
Sydney	17.829167

Average_temperature	
Month	
January	6.808333
February	7.158333
March	9.316667
April	11.375000
May	14.283333
June	16.758333
July	18.525000
August	18.675000
September	16.241667
October	12.825000
November	9.208333
December	7.166667



# Pivot table with aggregations: pivot\_table()

- What are the average minimum and maximum temperatures per city?

```
table = pd.pivot_table(df, values=['Minimum_temperature', 'Maximum_temperature'],
                       index=['City'], aggfunc={'Minimum_temperature':["mean"],
                       'Maximum_temperature':["mean"]})
```

City	Maximum_temperature	Minimum_temperature
	mean	mean
Amsterdam	13.416667	6.291667
Berlin	12.708333	6.083333
Brussels	14.758333	6.816667
London	15.083333	6.833333
Madrid	21.183333	9.508333
Sydney	21.725000	13.933333



# Pivot table with aggregations: pivot\_table()

- Could we add the sum for the minimum temperatures and the standard deviation for the maximum temperatures?

```
table = pd.pivot_table(df, values=['Minimum_temperature', 'Maximum_temperature'],
                       index=['City'], aggfunc={'Minimum_temperature': ["mean", "sum"],
                                              'Maximum_temperature': ["mean", "std"]})
```

	Maximum_temperature		Minimum_temperature	
	mean	std	mean	sum
City				
Amsterdam	13.416667	6.444989	6.291667	75.5
Berlin	12.708333	8.996106	6.083333	73.0
Brussels	14.758333	6.486554	6.816667	81.8
London	15.083333	5.869154	6.833333	82.0
Madrid	21.183333	8.306058	9.508333	114.1
Sydney	21.725000	3.760833	13.933333	167.2



# Crosstabulation

- Look at the data on the right
- We see some numbers (units and sales), but a lot of categorical data (date, region and type)
- What possible questions on grouped data might there be:
  - More children's clothing sold in the East?
  - More women's clothing sold in April?
  - Low sales in the South in winter months?
- Cross tabulation will help us investigate this

	Date	Region	Type	Units	Sales
0	11/07/2020	East	Children's Clothing	18.0	306
1	23/09/2020	North	Children's Clothing	14.0	448
2	2/04/2020	South	Women's Clothing	17.0	425
3	28/02/2020	East	Children's Clothing	26.0	832
4	19/03/2020	West	Women's Clothing	3.0	33
5	5/02/2020	North	Women's Clothing	33.0	627
6	24/01/2020	South	Women's Clothing	12.0	396
7	25/03/2020	East	Women's Clothing	29.0	609
8	3/01/2020	North	Children's Clothing	18.0	486
9	3/11/2020	East	Children's Clothing	34.0	374
10	16/04/2020	South	Women's Clothing	16.0	352
11	9/08/2020	North	Men's Clothing	NaN	270
12	1/05/2020	East	Men's Clothing	10.0	140
13	11/08/2020	East	Children's Clothing	12.0	348
14	7/01/2020	East	Men's Clothing	30.0	360



# Crosstab: region vs type

```
pd.crosstab(df.Region, df.Type)
```

- By default, in each cell the number of records in the DataFrame per region and type are shown

Type	Children's Clothing	Men's Clothing	Women's Clothing
Region			
East	113	122	176
North	85	89	142
South	45	39	53
West	42	41	53

- So, in each region the most sales registrations are for women's clothing.



# Crosstab: region vs type

```
pd.crosstab(index = df_pivot.Region, columns = df_pivot.Type,  
            values = df_pivot.Sales, aggfunc = 'mean')
```

- This is the same crosstab as on the previous slide, but now we're displaying the mean of sales revenue

- So now we know that:

- In women's clothing we have the lowest sales revenue.  
When women buy something, they buy less items or less expensive items. (From before we knew they made more sales though, so they might just like to go shopping.)
- Sales revenues are smaller in the East

Type	Children's Clothing	Men's Clothing	Women's Clothing
Region			
East	405.743363	423.647541	399.028409
North	438.894118	449.157303	432.528169
South	412.666667	475.435897	418.924528
West	480.523810	465.292683	419.188679



# An overview in graphs

- As people we like our numbers in graphs

- Unless you're him:



- When doing data science, which implies you have no or limited domain knowledge, you explore the data by making graphs
- These graphs raise questions, that spark more graphs
  - Repeat until you understand the data

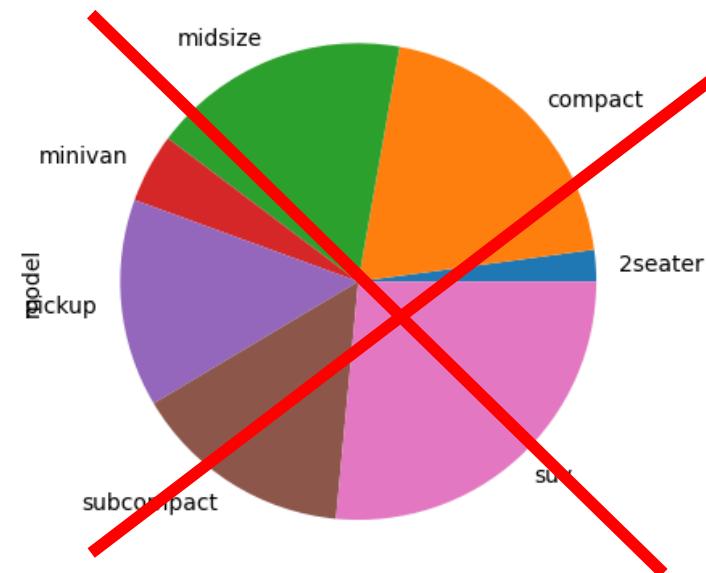


# The pie chart

- Pie charts are great. They not only have many colors, but they also remind us it's time for a break. We need more pie!
- But seriously, don't use pie charts unless you respect the rules:

- Max 6 slices
- In descending order
- Percentages printed

```
df.groupby("class").count().plot(kind="pie", y="model", legend=False)  
✓ 1.7s  
<AxesSubplot:ylabel='model'>
```





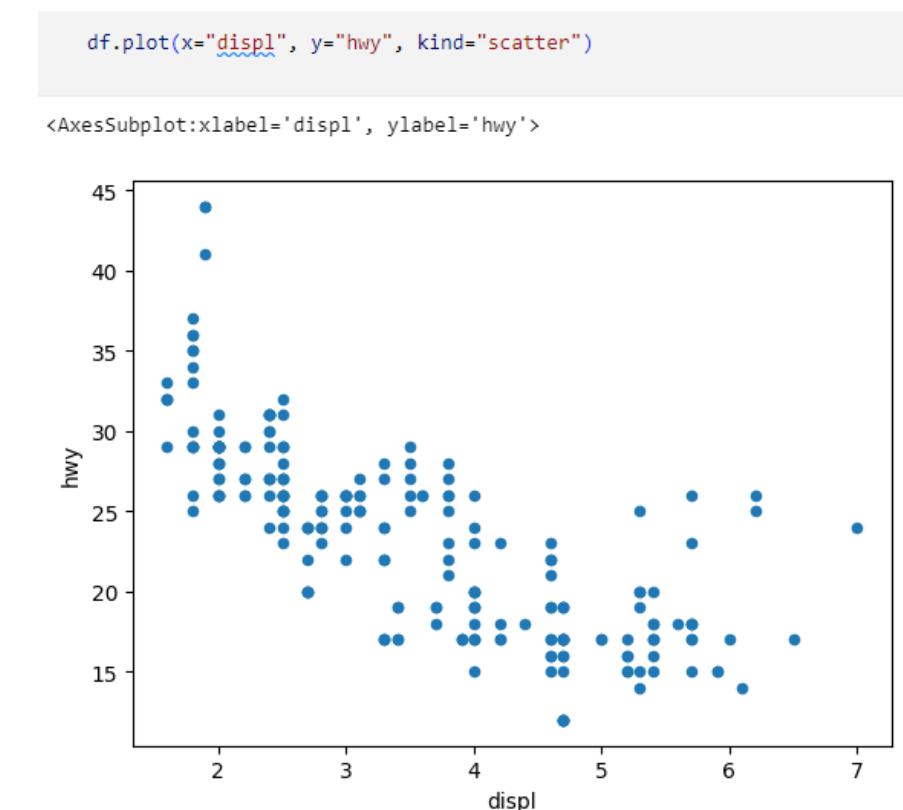
# Scatter plot

- Scatter plot:
  - Data visualisation technique to show the relationship between two numerical variables.
  - `DataFrame.plot.scatter(x, y, s = none, c = none)`
    - x: column name to be used as horizontal coordinates for each point
    - y: column name to be used as vertical coordinates for each point
    - s: size of dots
    - c: color of dots
- BTW, we're using the cars dataset 'MPG' here
  - The MPG dataset contains 234 cars and their mileage (~ chapter 5)
  - <https://ggplot2.tidyverse.org/reference/mpg.html>



# Scatter plot

- The easiest plot possible: plot highway miles per gallon vs engine displacement
  - How far you can drive vs how big your engine is

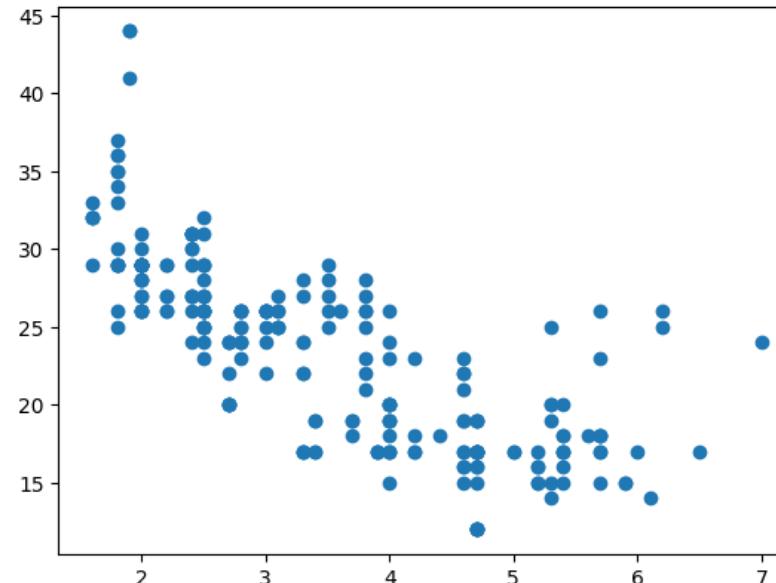




# Pandas vs matplotlib

- Pandas uses matplotlib under the hood, but you can use it separately
- You get the same graph, but the labels on the axes are gone

```
import matplotlib.pyplot as plt  
  
plt.scatter(df["displ"], df["hwy"])  
plt.show()
```





# Scatter with factors

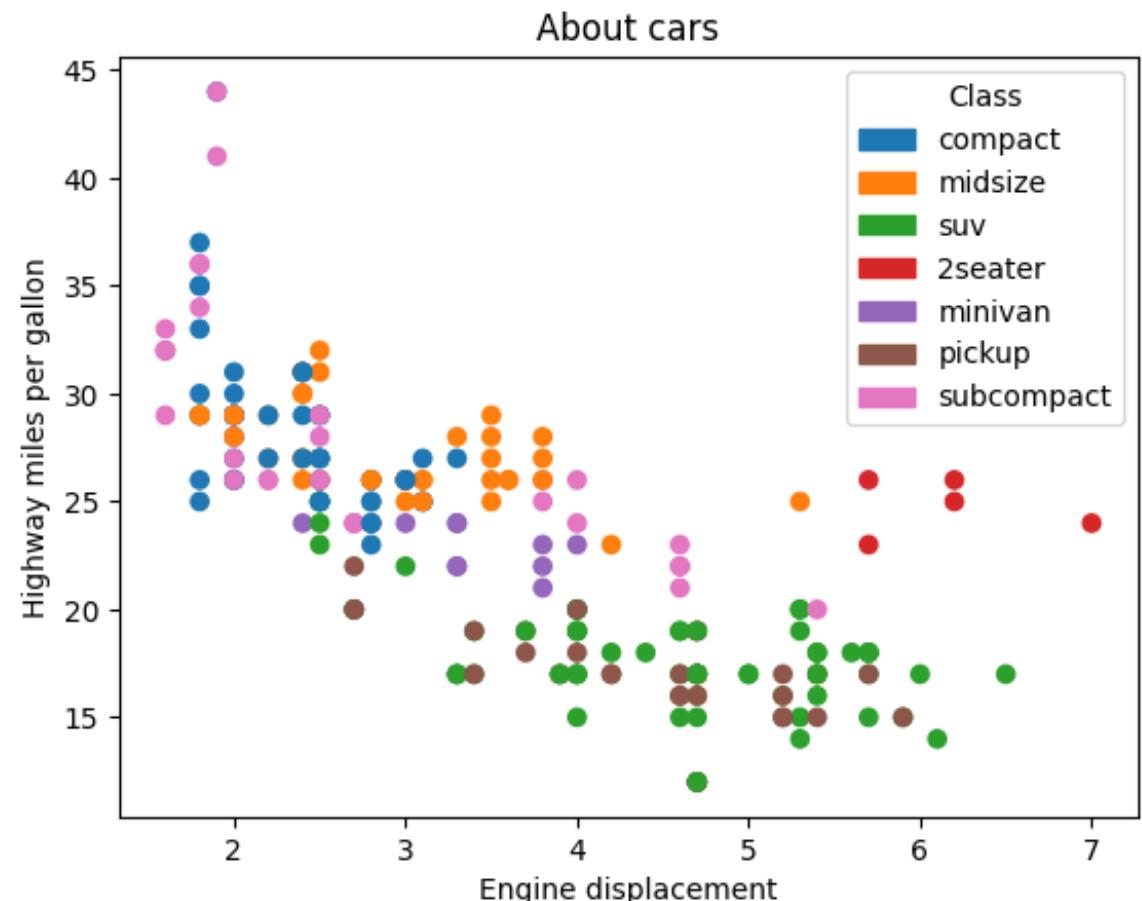
- Maybe the class of the car has an impact on this graph. Could we color the dots according to the class?

```
import matplotlib.patches

levels, categories = pd.factorize(df['class'])
colors = [plt.cm.tab10(i) for i in levels] # using the "tab10" colormap
handles = [matplotlib.patches.Patch(color=plt.cm.tab10(i), label=c) for i, c in enumerate(categories)]

plt.scatter(df.displ, df.hwy, c=colors)
plt.gca().set(xlabel='Engine displacement', ylabel='Highway miles per gallon', title='About cars')
plt.legend(handles=handles, title='Class')
```

Code also in notebook.





# A trendline: $ax+b$

- We've seen the trend in the data (bigger engines are less economical), but can we show it?

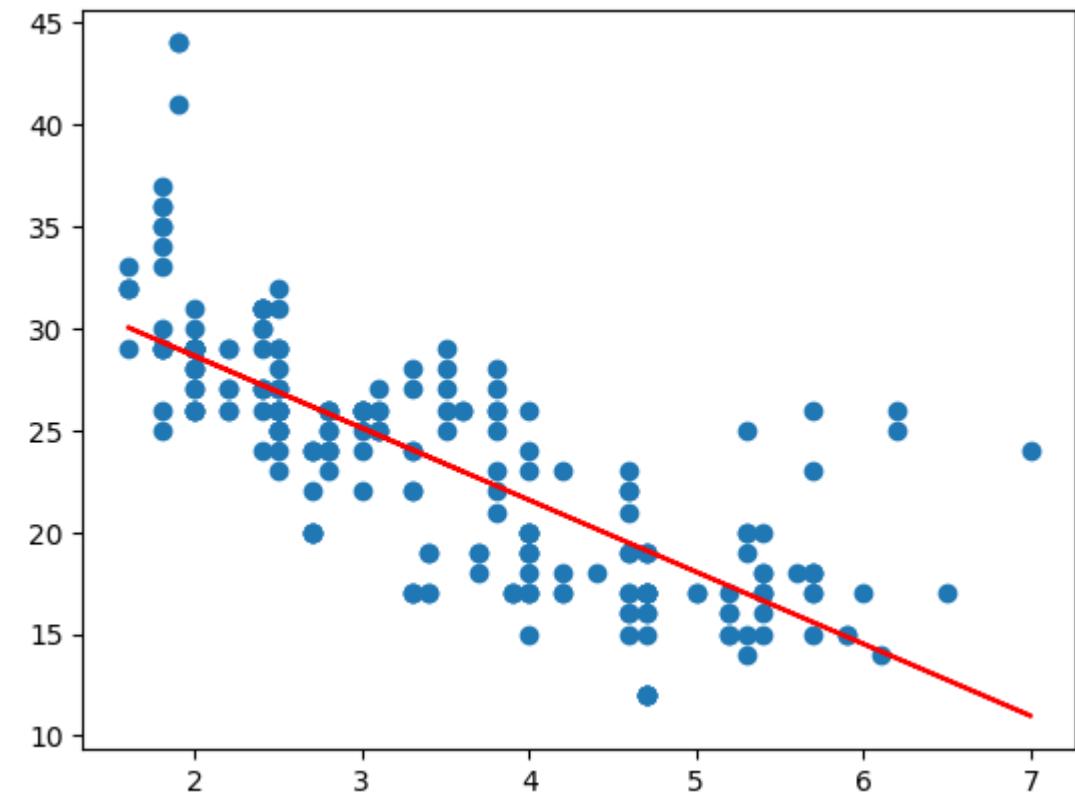
```
# https://www.statology.org/matplotlib-trendline/

import numpy as np

plt.scatter(df.displ, df.hwy)

#calculate equation for trendline
z = np.polyfit(df.displ, df.hwy, 1)
p = np.poly1d(z)

#add trendline to plot
plt.plot(df.displ, p(df.displ), color="red")
```





# A trendline: $ax^2+bx+c$

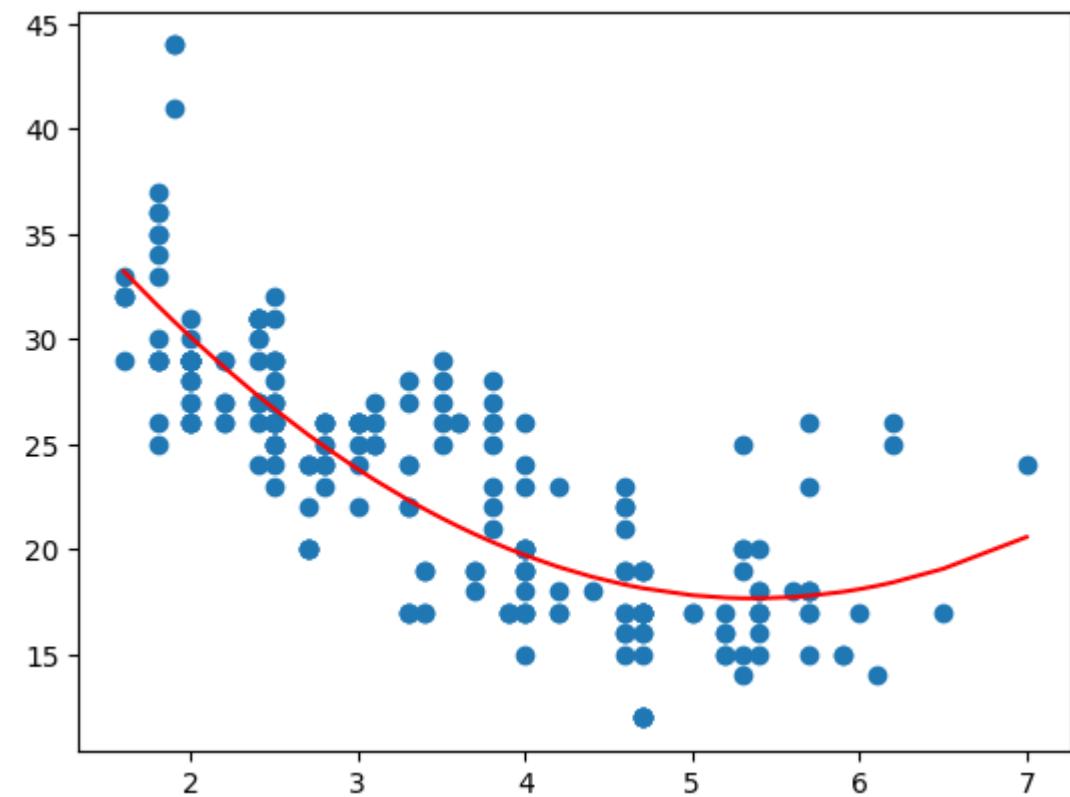
- Maybe the relationship is of the second order?

```
# https://www.statology.org/matplotlib-trendline/

plt.scatter(df.displ, df.hwy)

#calculate equation for trendline
z = np.polyfit(df.displ, df.hwy, 2)
p = np.poly1d(z)

#add trendline to plot
plt.plot(df.displ.sort_values(), p(df.displ.sort_values()), color="red")
```





# Trendline

- The trendline shows there is a relationship between engine displacement and highway miles per gallons
- In statistical terms, this is called a covariance
  - It's a number you can calculate by using some cool formula
  - But it doesn't mean anything, as it will greatly depend on the actual numbers
    - Big number: between number of ants and number of blades of grass
    - Small number: between number of kids and number of cars
- The strength of a covariance is expressed as the correlation
  - Can also be calculated using another cool formula
  - Is always between -1 and +1 -> **can be compared**

# Covariance and Correlation

Describes the relationship between two numerical variables



**Variable 1**

Temperature ( $^{\circ}\text{C}$ )



**Variable 2**

Ice-cream sales (\$)



Positive Covariance  
Positive Correlation



# Covariance and Correlation

Describes the relationship between two numerical variables



**Variable 1**

Temperature ( $^{\circ}\text{C}$ )



**Variable 2**

Pneumonia presentations (#)



Negative Covariance  
Negative Correlation





# The cool formulas

Day	x	y	x- $\bar{x}$	y- $\bar{y}$	(x- $\bar{x}$ )(y- $\bar{y}$ )
1	30	5	-3	-1	3
2	35	8	+2	+2	4
3	40	8	+7	+2	14
4	25	4	-8	-2	16
5	35	5	+2	-1	-2
Mean	33	6			Sum = 35

$$COV(x, y) = \sigma_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$
$$= \frac{35}{4} = 8.75$$



X and Y are positively related

Day	x	y	x- $\bar{x}$	y- $\bar{y}$	(x- $\bar{x}$ )(y- $\bar{y}$ )
1	30	5	-3	-1	3
2	35	8	+2	+2	4
3	40	8	+7	+2	14
4	25	4	-8	-2	16
5	35	5	+2	-1	-2
Mean	33	6			Sum = 35

$$COV(x, y) = \sigma_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$
$$VAR(x) = \sigma_x^2 = \frac{\sum(x_i - \bar{x})(x_i - \bar{x})}{n - 1}$$



# Correlation = measure of strength

Day	x	y	x- $\bar{x}$	y- $\bar{y}$	(x- $\bar{x}$ )(y- $\bar{y}$ )
1	30	5	-3	-1	3
2	35	8	+2	+2	4
3	40	8	+7	+2	14
4	25	4	-8	-2	16
5	35	5	+2	-1	-2
Mean		33	6	Sum = 35	

$$\text{CORR}(x, y) = \rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$
$$= \frac{8.75}{(5.70)(1.87)} = 0.820$$

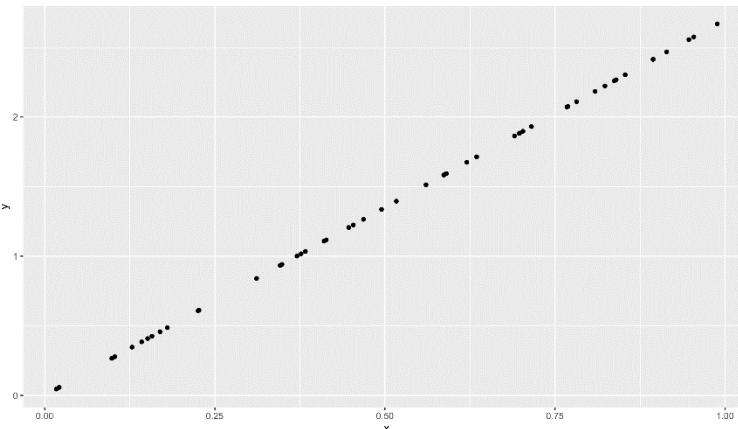
X and Y are  
strongly positively  
related



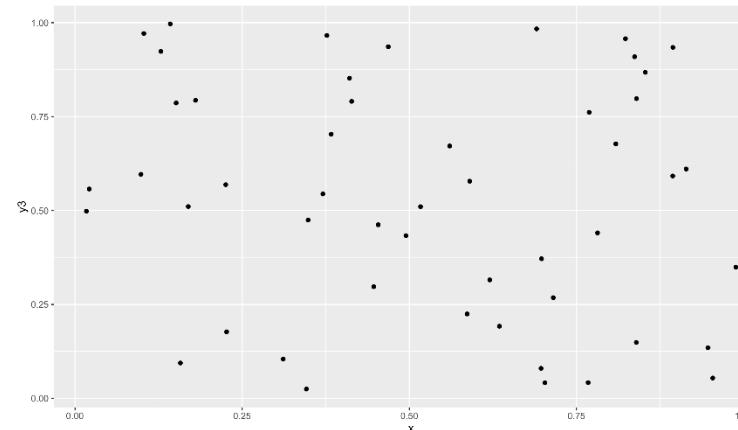
$$-1 \leq \rho \leq 1$$



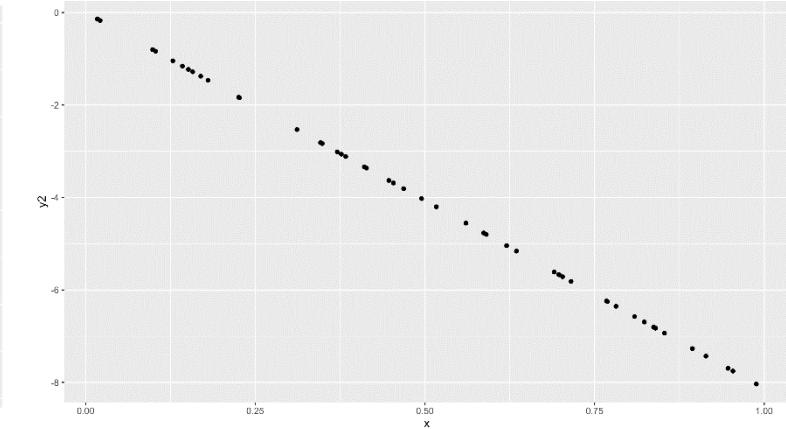
# Extremes of correlation



**Correlation = 1**  
Perfect positive relationship



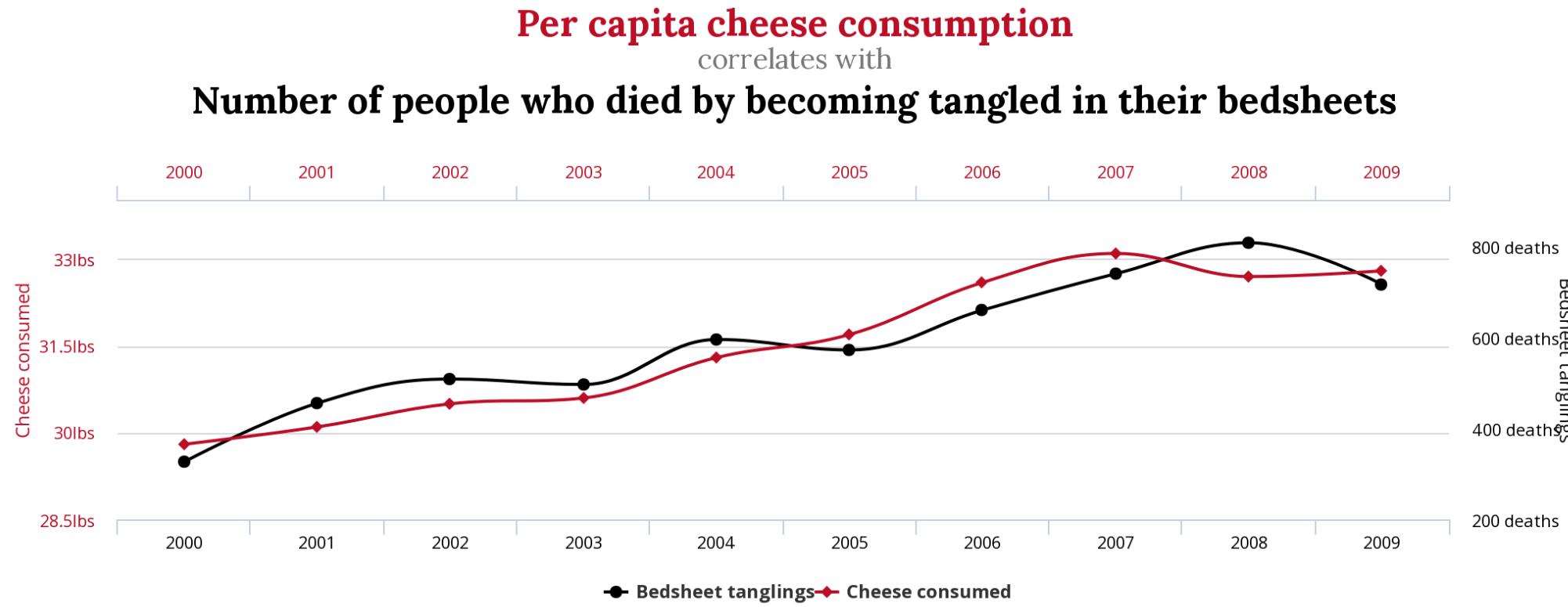
**Correlation = 0**  
No pattern



**Correlation = -1**  
Perfect negative relationship



# Covariance does not imply causation



tylervigen.com



# Correlation in DataFrames

- A scatter plot is a very useful plot to check if there is correlation between two variables in a dataset
- But which variables correlate well, and which don't? You have 2 options:
  - Check numerically: calculate correlation matrix
  - Check visually: plot a scatter matrix



# Correlation matrix

- Calculate the correlation between all columns
  - Or all numerical columns, to be precise
- Gives you a number showing how much they are related
  - 0: No correlation
  - 1: One goes up, other goes up
  - -1: One goes up, other goes down
- The closer the number is to +1/-1, the better the correlation

```
# pairwise correlation
corr = df[['displ', 'year', 'cyl', 'cty', 'hwy']].corr()
corr.style.background_gradient(cmap = 'coolwarm')
```

✓ 0.0s

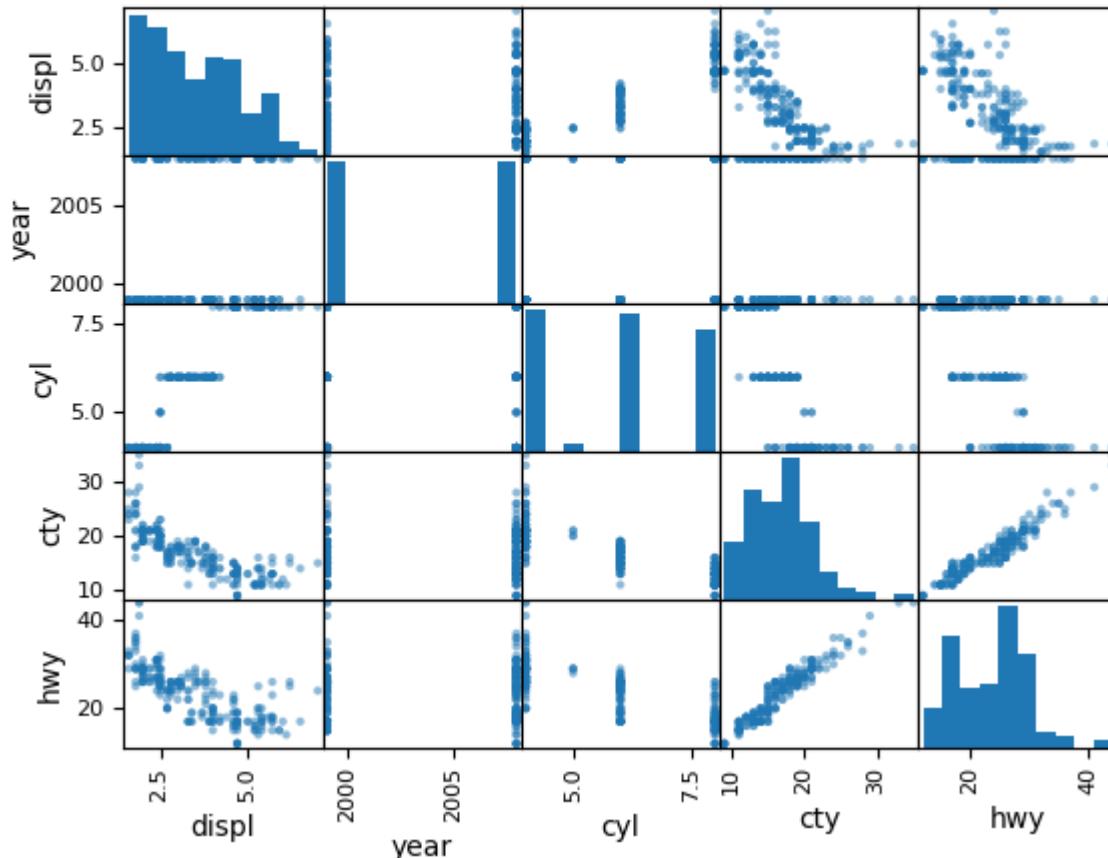
	displ	year	cyl	cty	hwy
displ	1.000000	0.147843	0.930227	-0.798524	-0.766020
year	0.147843	1.000000	0.122245	-0.037232	0.002158
cyl	0.930227	0.122245	1.000000	-0.805771	-0.761912
cty	-0.798524	-0.037232	-0.805771	1.000000	0.955916
hwy	-0.766020	0.002158	-0.761912	0.955916	1.000000



# Scatter matrix

- The same as a correlation matrix, but instead of numbers you get a scatter plot
- Some plots stand out:
  - cty vs hwy: near perfect correlation
  - year vs ...: Only 2 distinct values for year are present
  - cyl: few distinct values, but some correlation with displ

```
from pandas.plotting import scatter_matrix
null = scatter_matrix(df)
✓ 0.8s
```





# Scatter matrix

---

- Did we see that correctly?
- cty vs hwy: near perfect correlation
  - cty: Miles per gallon in the city
  - hwy: Miles per gallon on the highway
- cyl: few distinct values, but some correlation with displ
  - cyl: the number of cylinders
  - displ: the size of the engine (in liters)
- So yes, what we saw in the graphs is also true in real life



# Boxplots, skewing and distributions

- When investigating it's important to recognize certain patterns in the distribution of data
- Fortunately, there's a whole field of knowledge concerning itself with just that:

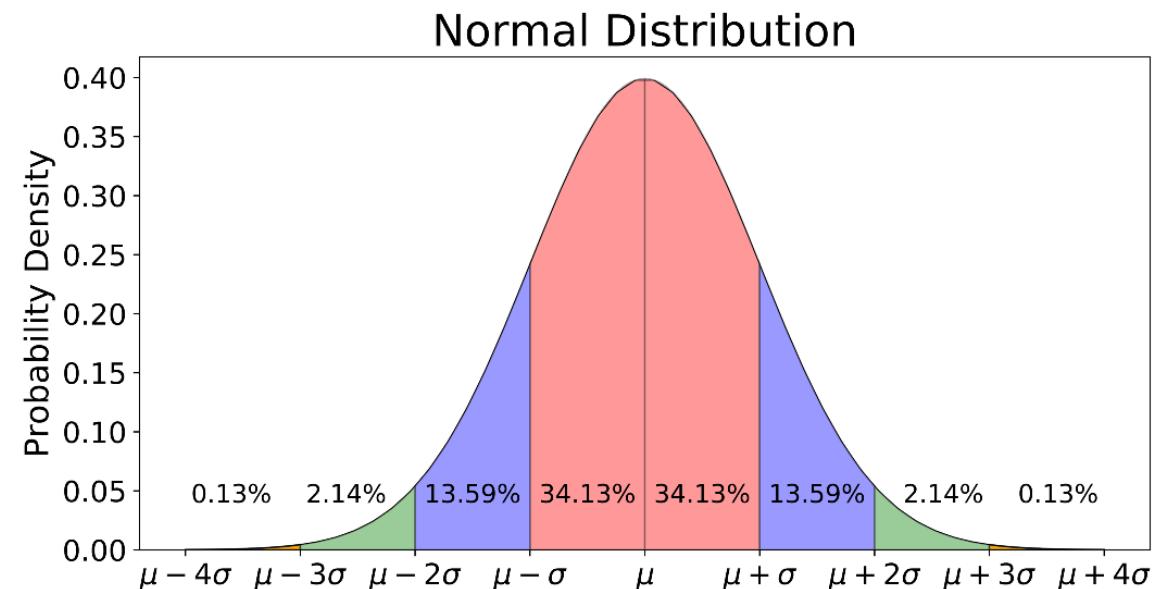
## Statistics!

- And we'd like very much to prep you for a couple of weeks in just the theoretical background of analysis of distributions, correlations, covariations, ...
  - But that would be a step too far...
- But what you do need to understand is the **normal distribution**



# Normal distribution

- Most data will have a “normal” distribution
- This means:
  - A lot of datapoints in the center (red part)
  - Some datapoints to the side (blue part)
  - Leftovers left and right (green and ... part)
- Standard deviation is a measure for the steepness of the curve

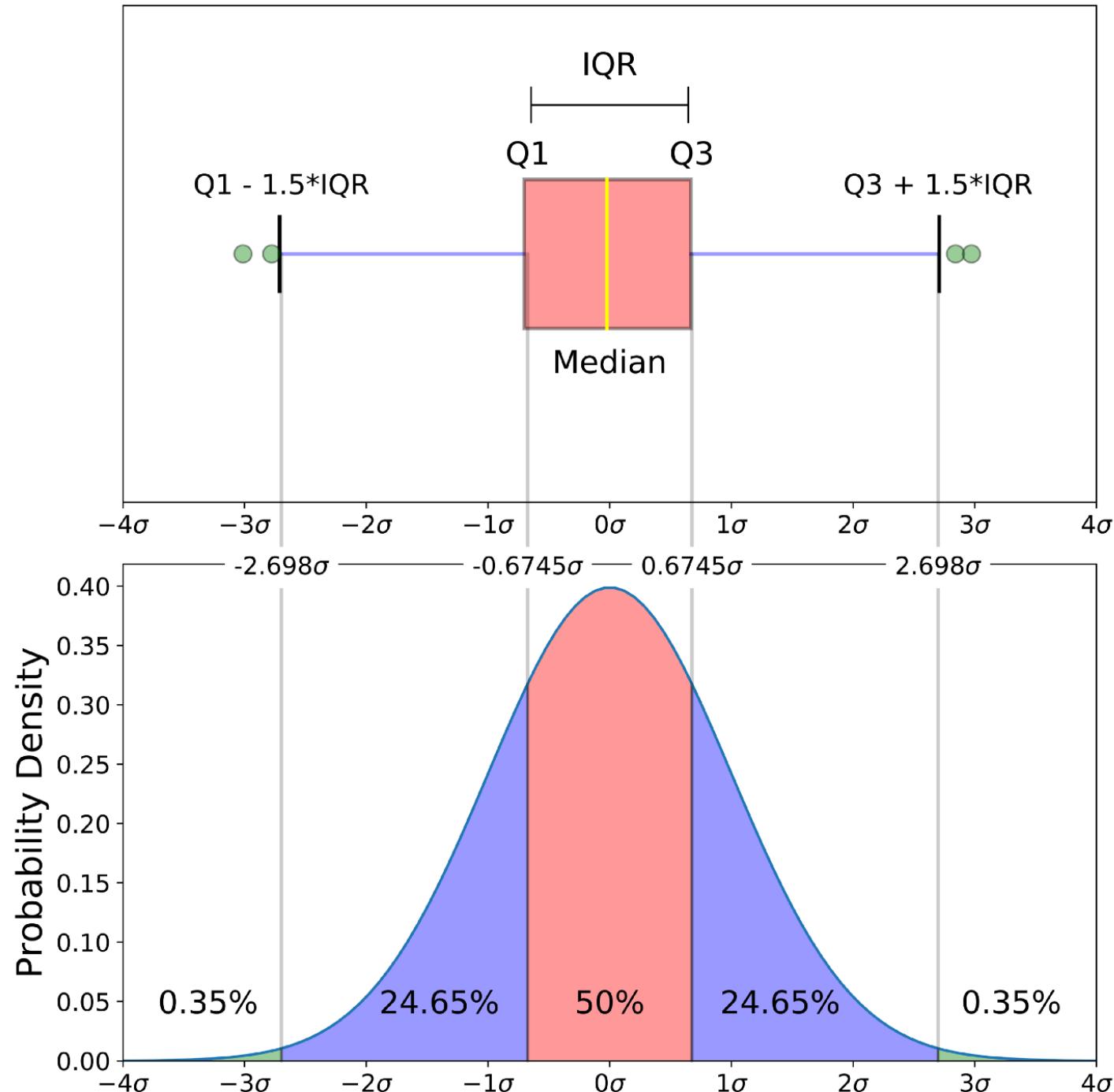




# Normal distribution in boxplot

- A normal distribution is a plot of 1 variable, with varying standard deviation
  - The smaller your stddev, the steeper the curve, the easier to predict
  - The bigger your stddev, the flatter and widen the curve, the more spread your data, the more difficult to predict
- In case you want to compare 2 variables like prices of fiction vs prices of non-fiction books, you need a more compact way of showing a distribution
- Enter: The Boxplot

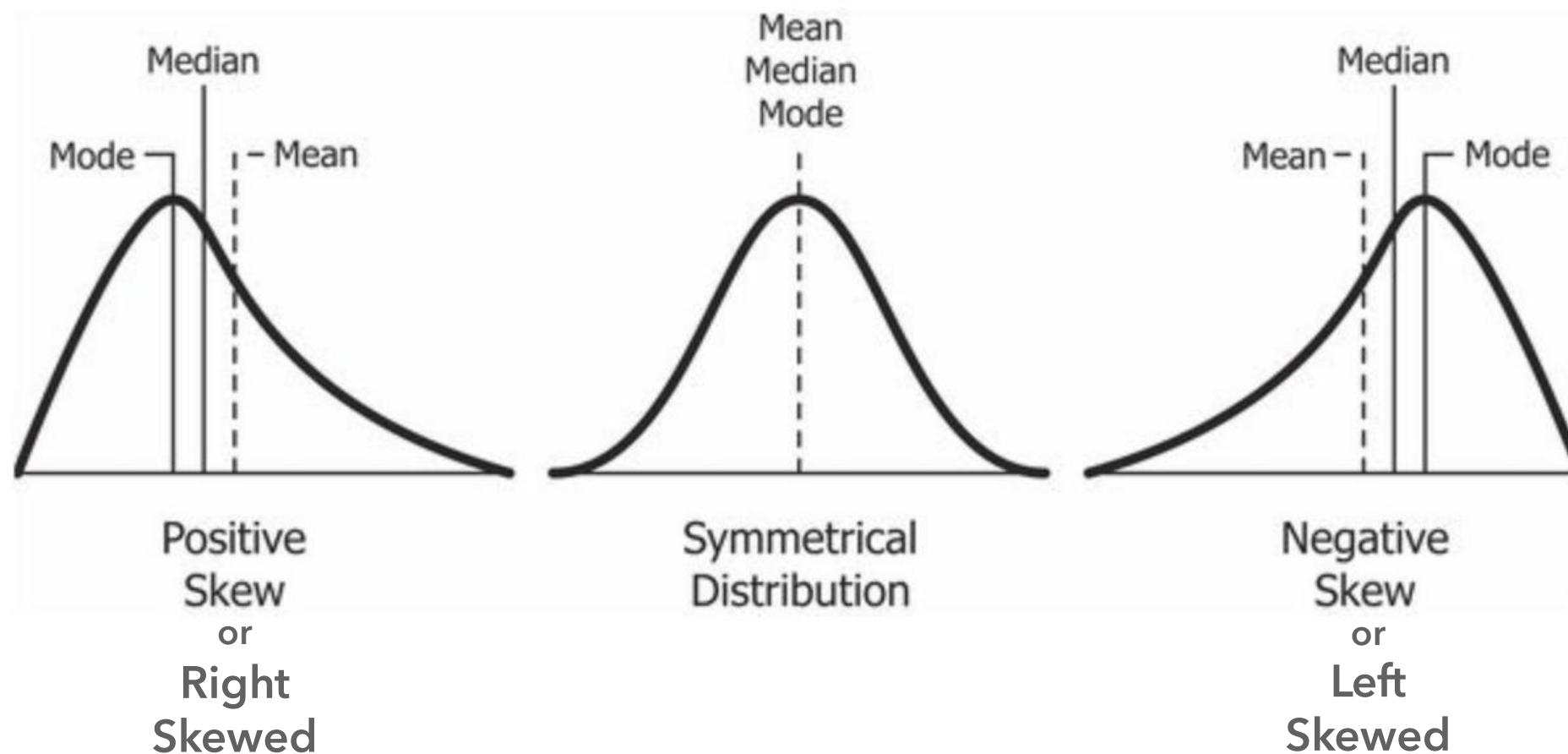
- The yellow line: the median
- The box: the red part
- The lines: the blue part
- The dots: the green part
- IQR: Interquartile Range  
The difference between Q1 and Q3 which measures the spread of the middle 50% of the data.





# Dataset skewing

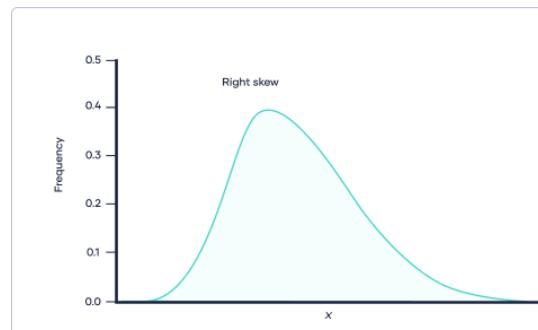
- All data is ‘normal’, but some is more normal than others



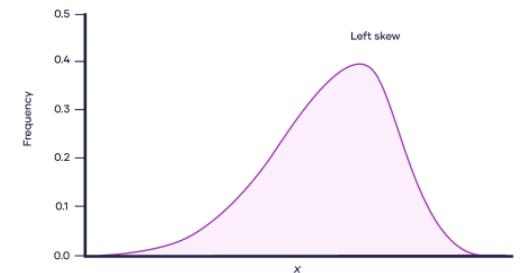
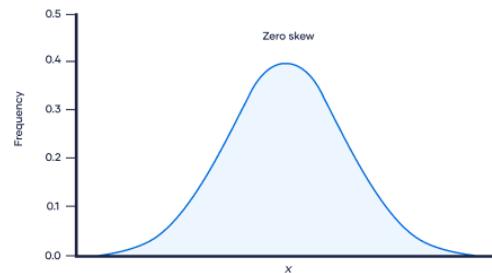


# skew()

- **Skewness** is a measure of the asymmetry of a distribution. A distribution is asymmetrical when its left and right side are not mirror images.
- Describes the distribution of a variable alongside other descriptive statistics
- Determines if a variable is normally distributed. A normal distribution has zero skew and is an assumption of many statistical procedures.



<https://www.scribbr.com/statistics/skewness/>

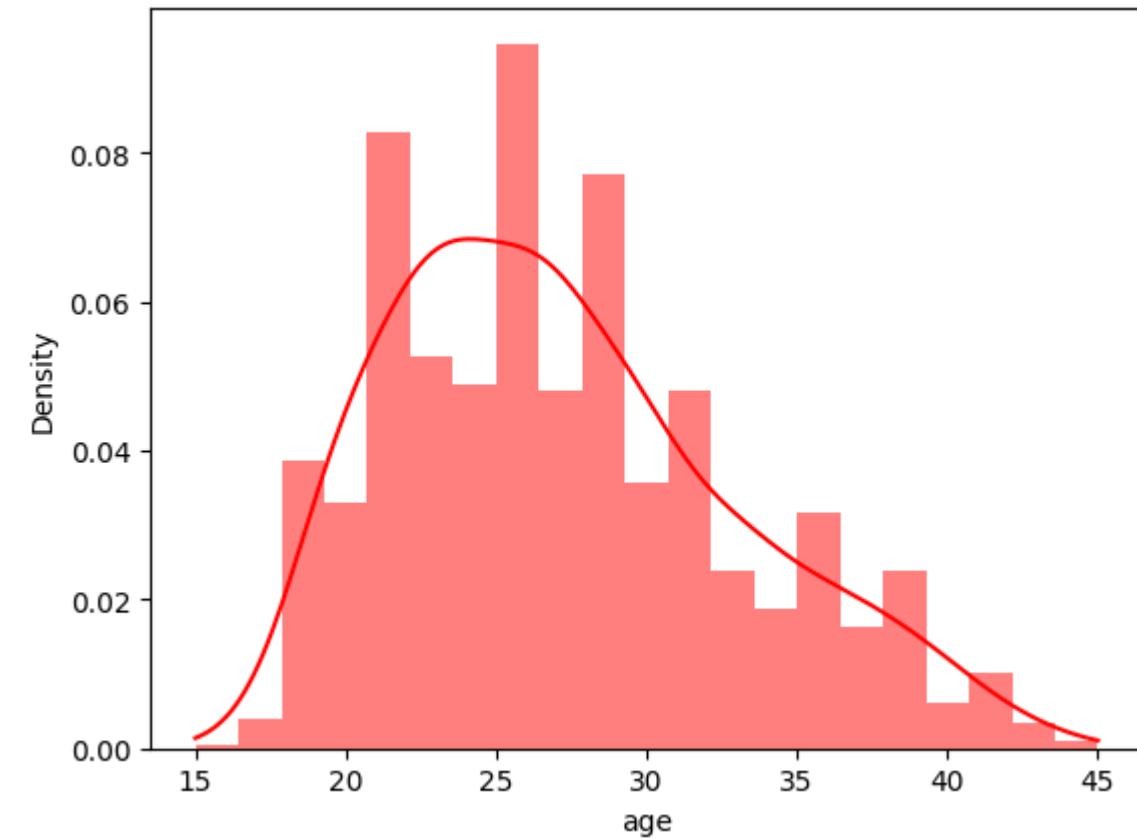


```
1 baby.skew()
✓ 0.5s
case      0.000000
bwt       -0.140815
gestation -0.778776
parity     1.126458
age        0.584561
height    -0.086639
weight     1.259159
smoke      0.431049
dtype: float64
```



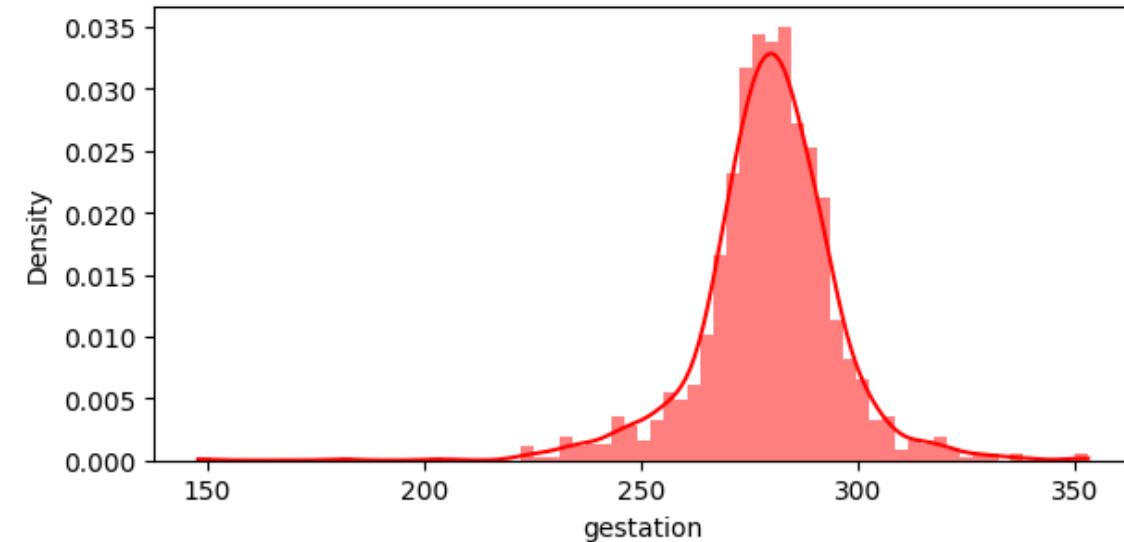
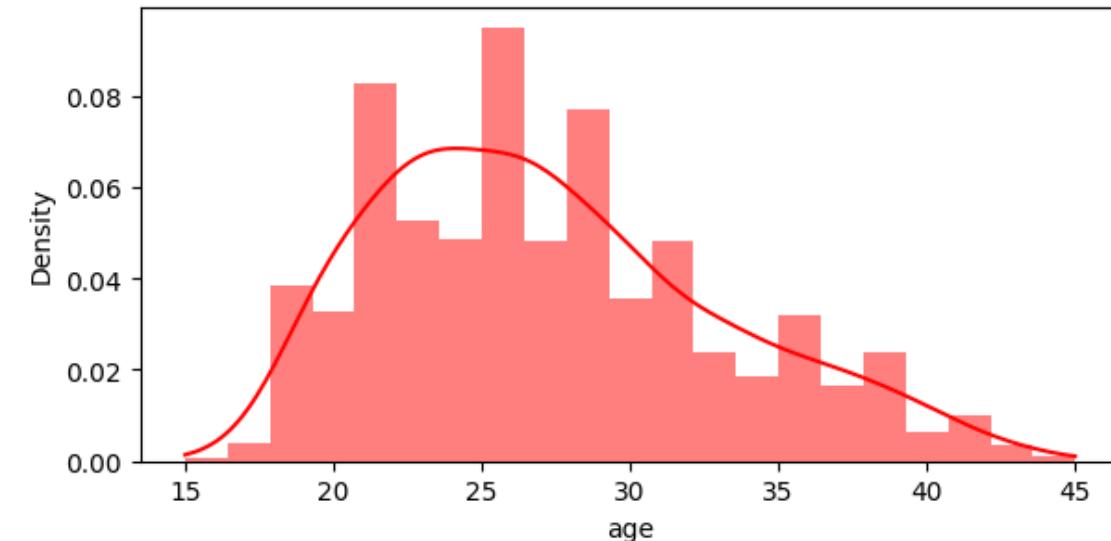
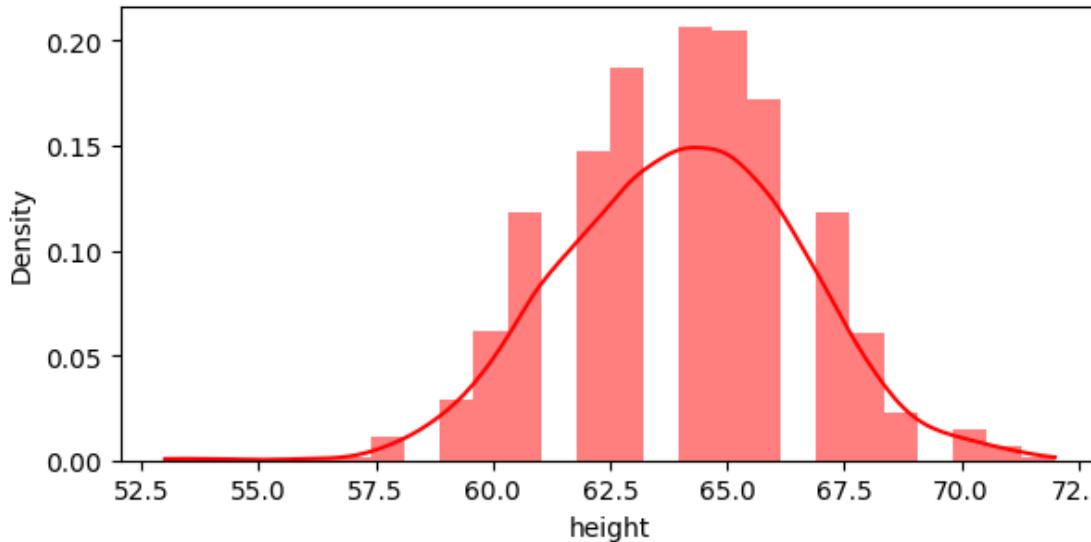
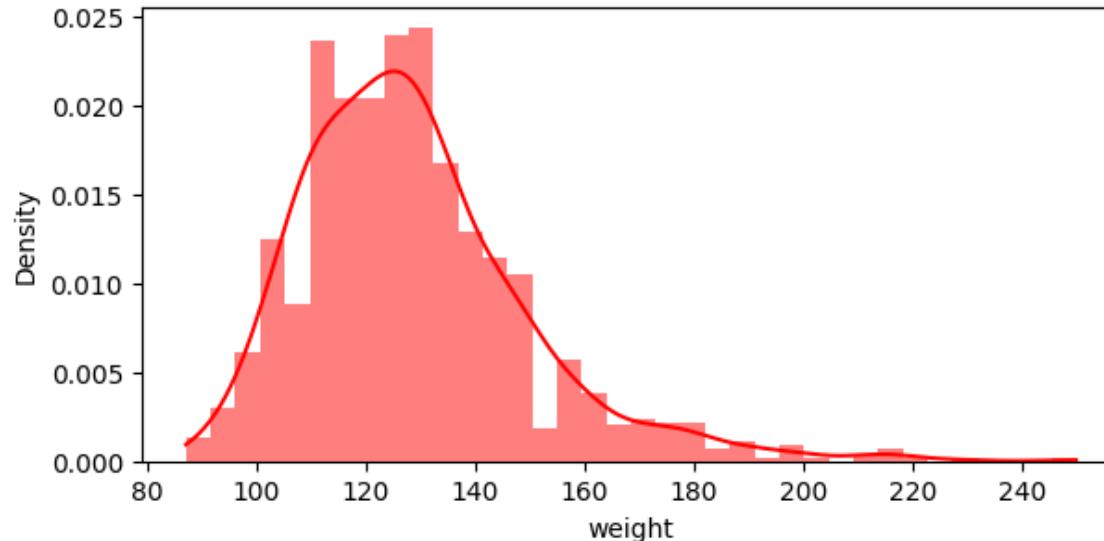
# Example: age of mother when giving birth

- Some mothers are only 15, but these are exceptions
- Most women give birth in their 20's
  - School is finished, found life-long partner, ...
- There are still quite a few women giving birth in their 30's
  - Enjoyed life first, settled later
- Some exceptions are in their 40's
  - You'll be 60 by the time the kid graduates, which is not ideal
  - Conception becomes difficult after that age





# More baby-examples





# Negative skew or left skewed

- Distribution of Age of Deaths
  - The distribution of the age of deaths in most populations is negatively skewed. Most people live to be between 70 and 80 years old, with fewer and fewer living less than this age.
- Distribution of Olympic Long Jumps
  - In most years, the distribution of long jump lengths for competitors in the Olympics is negatively skewed because most competitors land a jump around 7.5-8 meters, with a few landing a jump of just 5-6 meters.
- Distribution of Scores on Easy Exams
  - The distribution of scores on easy exams or tests tend to be negatively skewed because most students score very high, while a few students score much lower than the average.



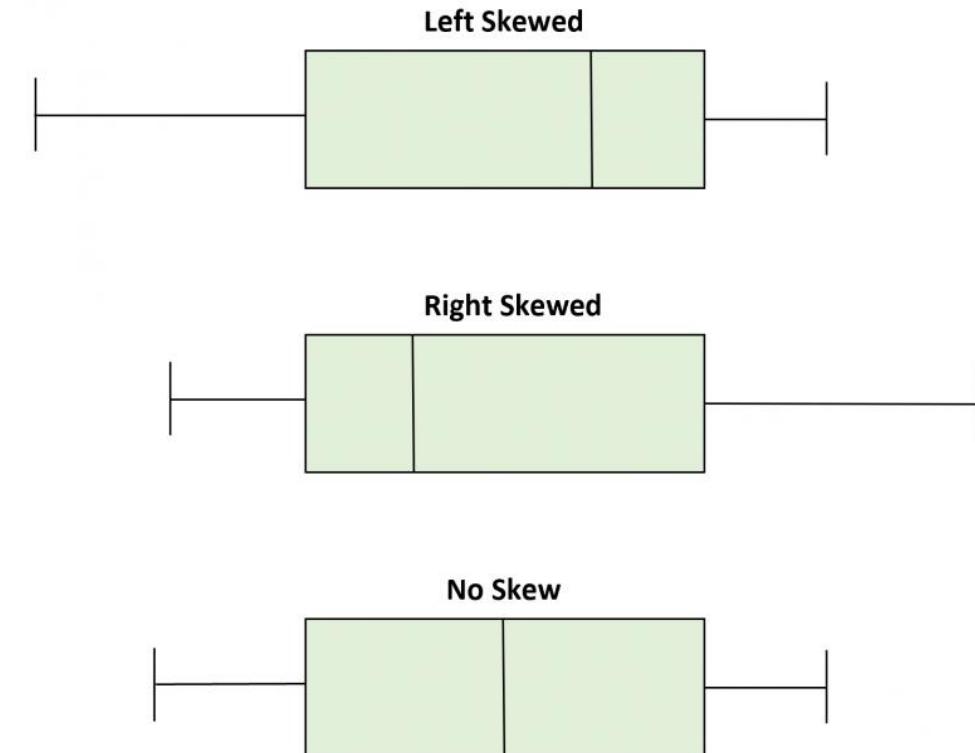
# Positive skew or right skewed

- Distribution of Income
  - The distribution of individual incomes in the U.S. is right-skewed, with most individuals earning between \$20k and \$40k per year but with a long right tail of households that earn much more.
- Distribution of Pet Ownership
  - The distribution of the number of pets that households own in any particular city is likely to be right skewed because most households have either 0 or 1 pet, but there are many outlier households that have 7, 8, 9+ pets that cause the distribution to be right skewed.
- Distribution of Scores on Difficult Exams
  - The distribution of scores on any particularly difficult exam will be positively skewed with most students scoring around some mean value with a few outlier students scoring much higher.



# Boxplot skewing

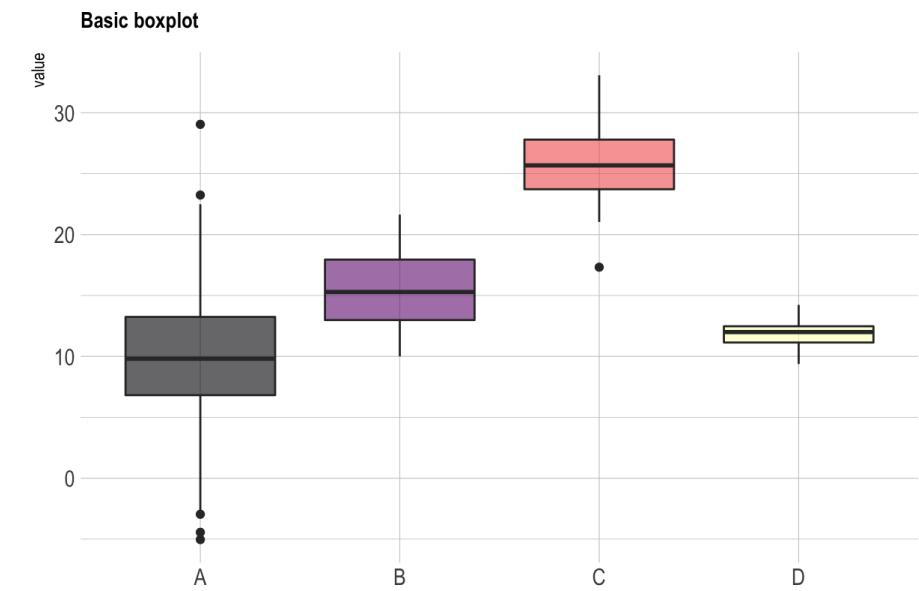
- How does a boxplot show skewing?
- The middle line is always the center of your data (median)
- If that is a perfect mirror-line, your data is perfectly symmetrical and perfectly normal
- But if the box is smaller on one side of the line, the data is “steeper” there
  - Less different values, but they occur more
- Usually, a small box on one side also means a shorter line





# Basic boxplot

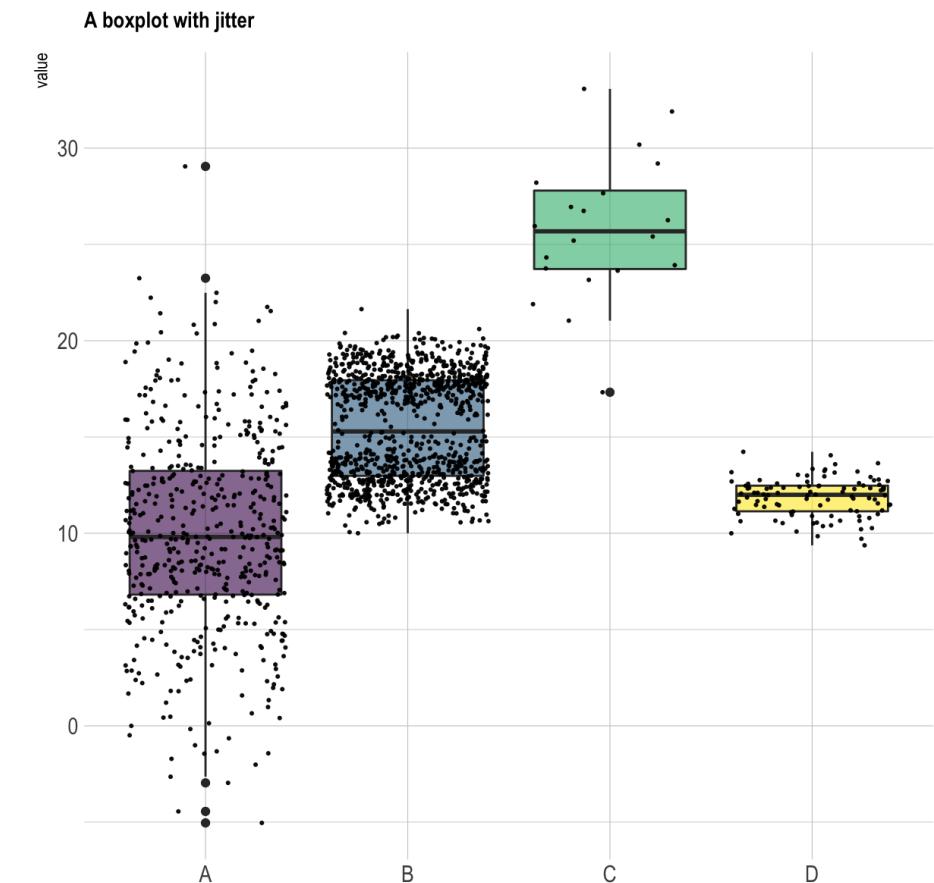
- Compare the distribution of 4 categories
  - 1 variable
  - In 4 categories
- You see how
  - A is very spread out (unsteep bell curve)
  - D is very compact (pointy bell curve)
  - All are pretty ‘normal’ or symmetrical distributed





# Advanced boxplot

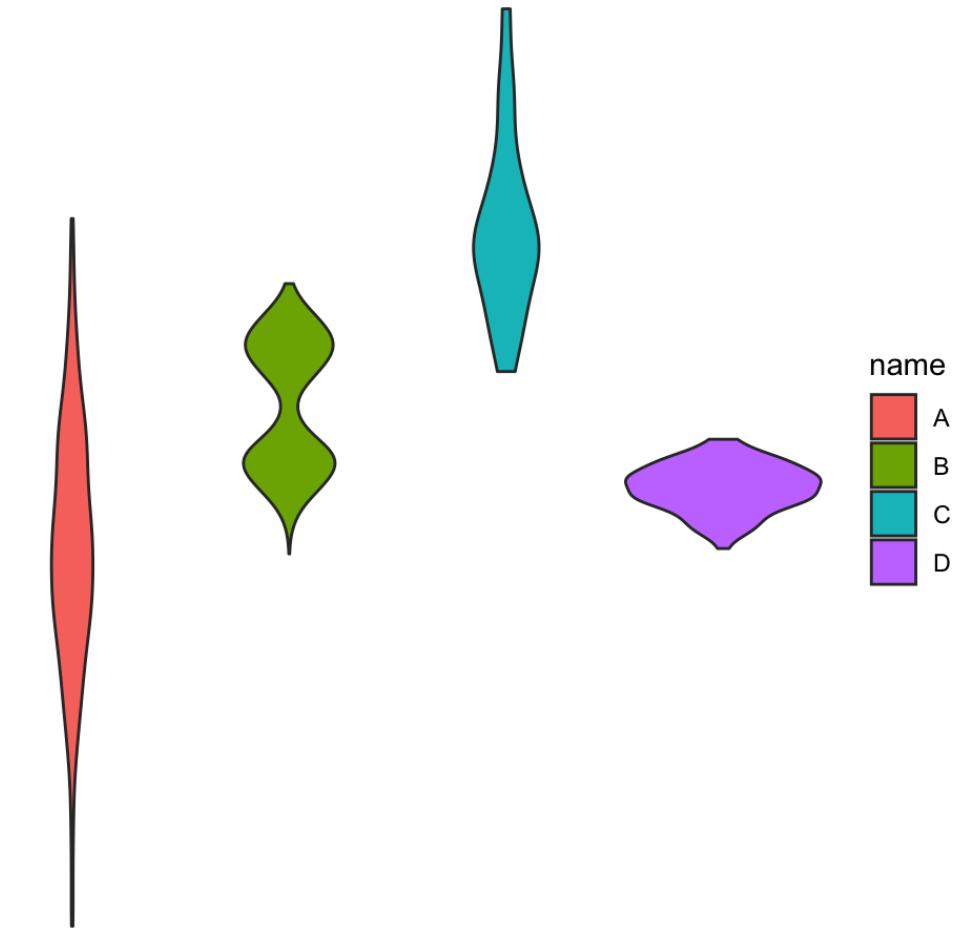
- You see all the above and:
- C doesn't have a lot of datapoints
- B has two peaks





# Violin plot

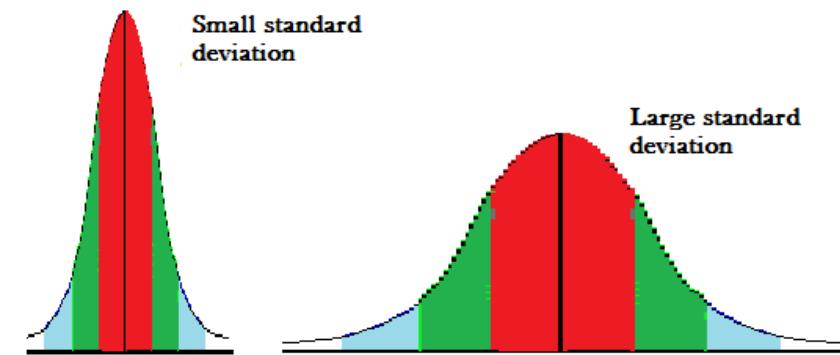
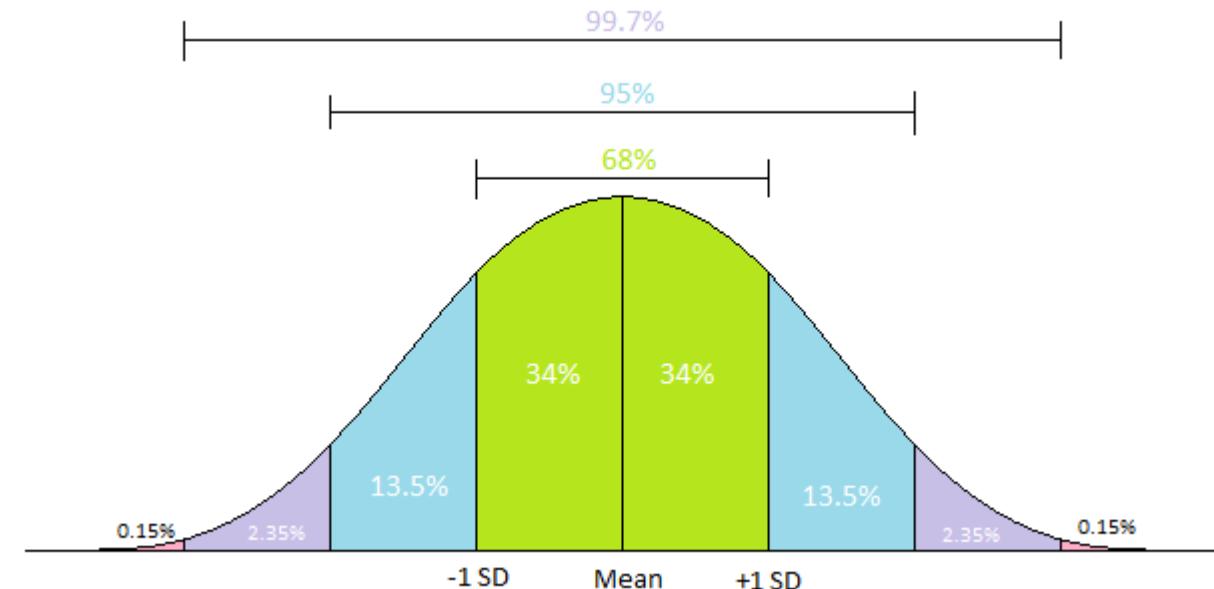
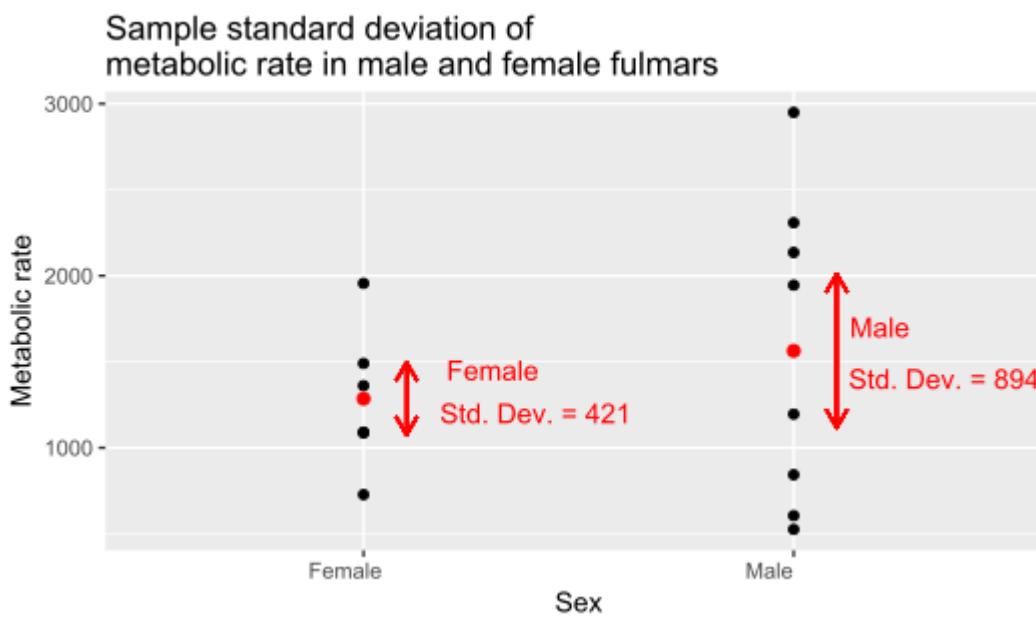
- Comparable to a boxplot, but the numbers are no longer there
- Prettier, but not always better
  - Because it doesn't compare: are there more A's than D's? Or C's? How much steeper is A's curve compared to C?
  - You can use a violin-plot to raise questions, but rarely to answer them
- Example: is there a property in the B-dataset that splits the two bumps?





# Once again: Mean (average), standard deviation

Population Mean	Sample Mean
$\mu = \frac{\sum_{i=1}^N x_i}{N}$	$\bar{X} = \frac{\sum_{i=1}^n x_i}{n}$
$N$ = number of items in the population	$n$ = number of items in the sample





# Mean (average), standard deviation

## Variance and standard deviation (actually) explained

Week	Weekly expenditure on Golden Gaytimes
1	\$48.50
2	\$87.40
3	\$19.98
4	\$59.74
5	\$40.87
6	\$105.51
7	\$40.80
8	\$23.10
9	\$98.10
10	\$60.54
11	\$64.81
12	\$48.01



$$\text{Mean} = \bar{x} = \frac{\sum x}{n} = \frac{\$48.50 + \$87.40 + \dots}{12} = \$58.11$$

$$\text{Variance} = s^2 = \frac{\sum(x - \bar{x})^2}{n - 1} = \frac{(\$48.50 - \$58.11)^2 + (\$87.40 - \$58.11)^2 \dots}{11} = \$748.01$$

$$\text{Std dev} = s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}} = \sqrt{\$748.01} = \$27.35$$

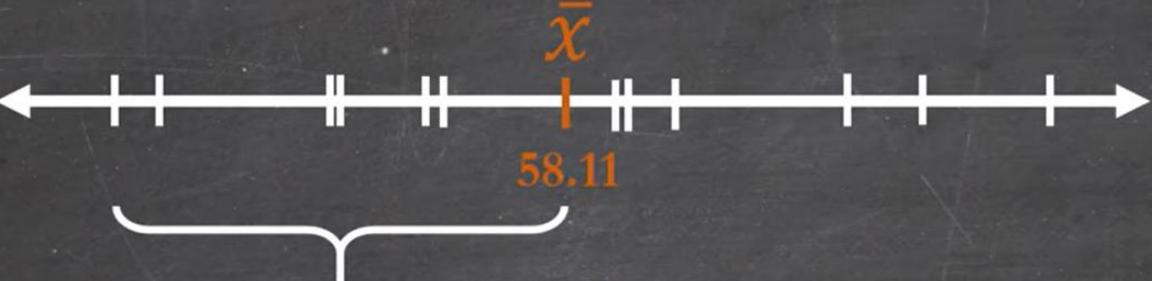


# Mean (average), standard deviation

 Why do we bother with "variance"?  
(ie. why square stuff?)

**Objective:** describe the spread of the data

Lowest observation: \$19.98  
Highest observation: \$105.51



A horizontal number line with tick marks. The mean is marked with a vertical line and labeled  $\bar{x}$ . The value 58.11 is also labeled below the mean. A bracket below the number line spans from the lowest observation to the highest observation.

Squared deviation =  $(-38.13)^2 = 1453.897$

**Second thought:** find the average **squared** deviation from the mean



# Mean (average), standard deviation

- Why square stuff for calculating variance and standard deviation?

Raw data	Deviation from mean	Squared deviation
15	$15 - 9.5 = 5.5$	30.25
3	$3 - 9.5 = -6.5$	42.25
12	$12 - 9.5 = 2.5$	6.25
0	$0 - 9.5 = -9.5$	90.25
24	$24 - 9.5 = 14.5$	210.25
3	$3 - 9.5 = -6.5$	42.25
$M = 9.5$		Sum = 0
		Sum of squares = 421.5

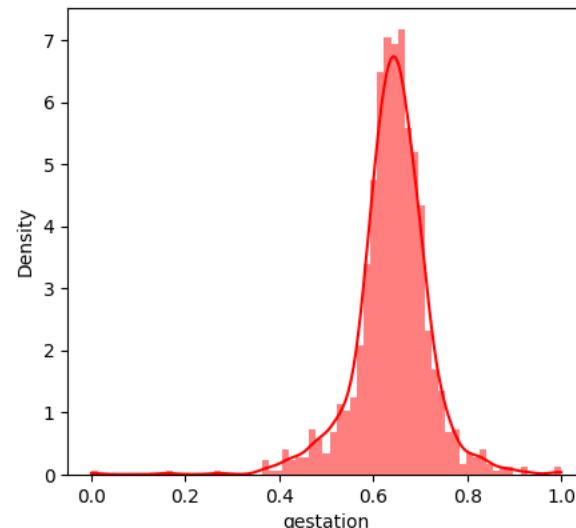
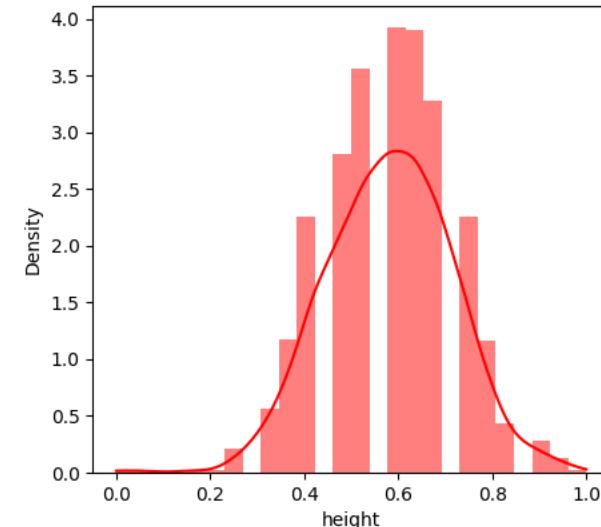
1. Add up all of the squared deviations:  
 $\rightarrow 421.5$
2. **Variance ( $s^2$ )**  
Divide the sum of the squared deviations by  $N - 1$ :  
 $\rightarrow 421.5 / (6-1) = 84.3$
3. **Standard deviation ( $s$ )**  
Calculate the square root of variance  
 $\rightarrow \sqrt{84.3} = 9.18$

From learning that  $s = 9.18$ , you can say that on average, each score deviates from the mean by 9.18 points.



# Back to the babies

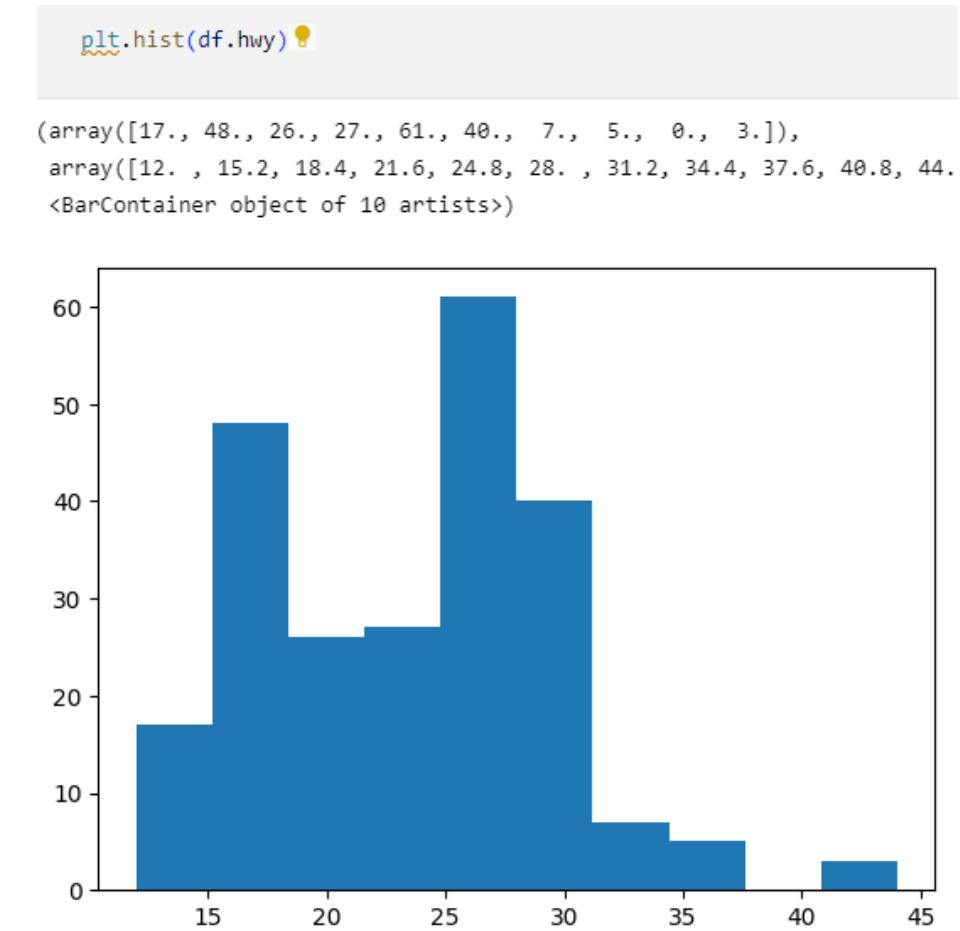
- On the right are 2 graphs: the height of a baby at birth and the duration of the pregnancy (gestation)
- To be comparable they were normalised
  - Recalculated to be between 0 and 1 instead of 150 and 350 or 52 and 72
  - [For more information on normalisation see chapter 4]
- The top graph is wider, the bottom one slimmer
  - Top standard deviation: 0.13
  - Bottom standard deviation: 0.08





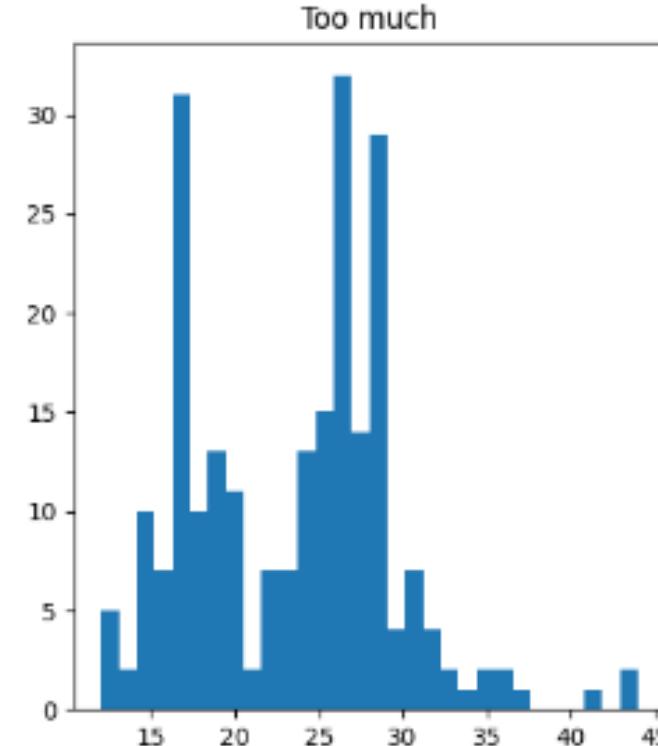
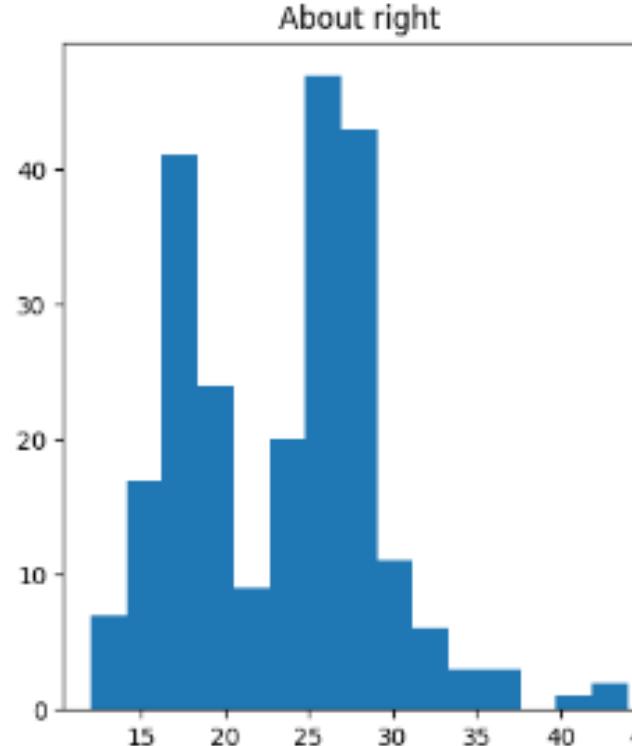
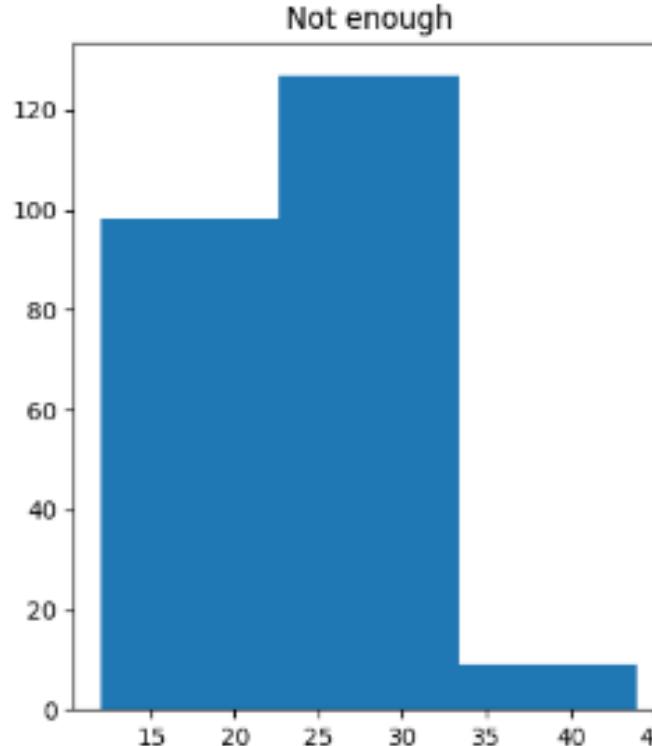
# Distributions in mpg

- Before we go into deep analysis, let's start practically:
- Create a simple plot of the distribution for highway miles per gallons in the MPG-dataset
- It's a continuous variable, so we can't count it
- Solution: binning
  - Take some of the values together
  - Which? Size of the bin
  - In this case: let pandas decide on the bin size



# Binning

- Maybe more bins? Or less bins?
  - Depends on the data...



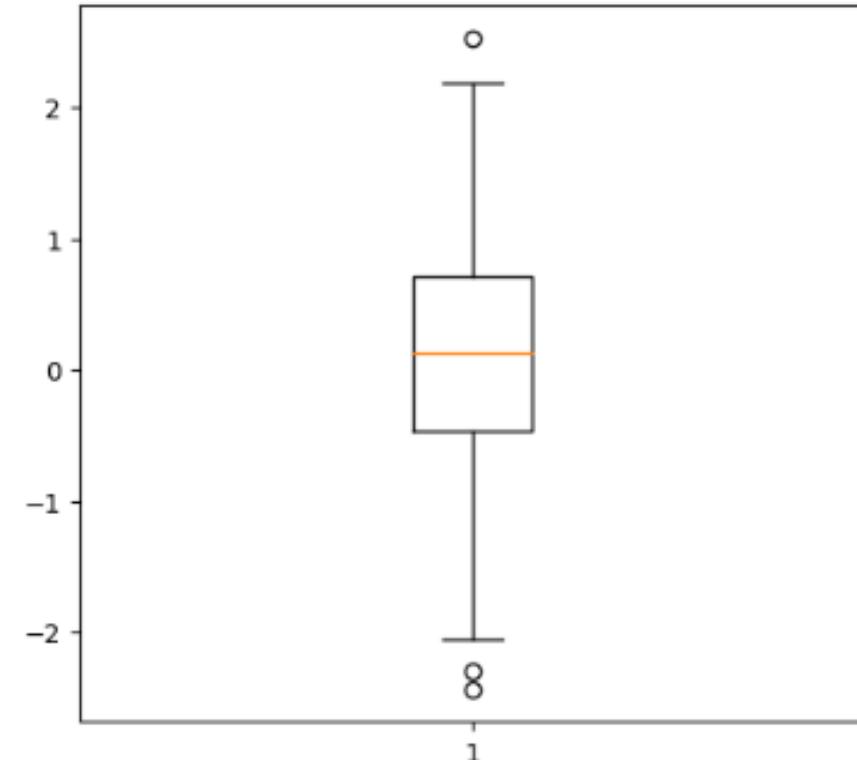
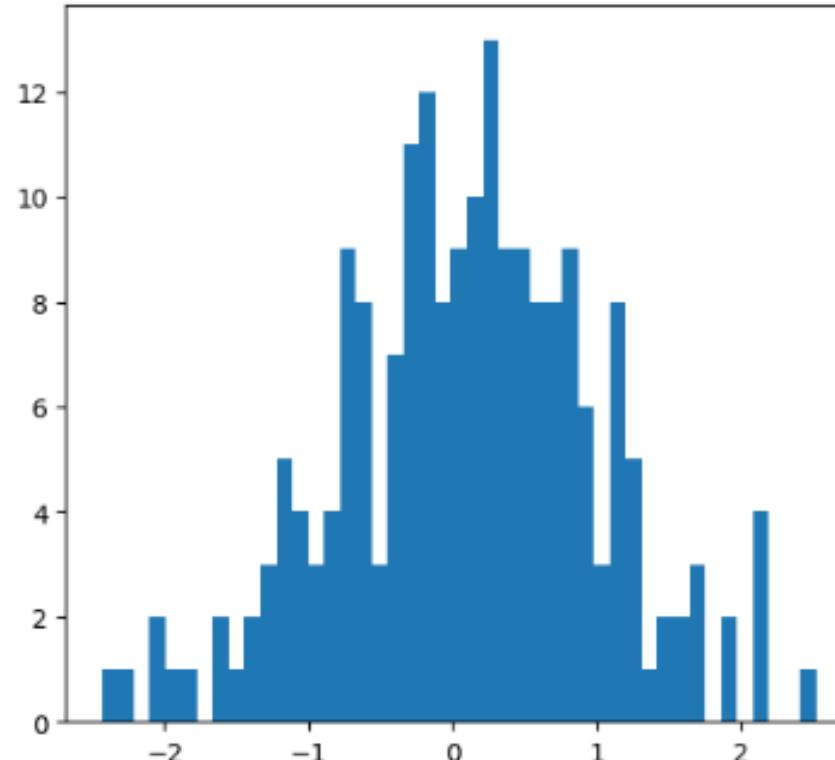
```
figure, axis = plt.subplots(1, 3, figsize=(15,5))
# plt.figure(figsize=(1, 3))

axis[0].hist(df.hwy, bins=3)
axis[0].title.set_text("Not enough")
axis[1].hist(df.hwy, bins=15)
axis[1].title.set_text("About right")
axis[2].hist(df.hwy, bins=30)
axis[2].title.set_text("Too much")

plt.show()
```

# Truly normal data

- Mpg doesn't have truly normal data (doesn't have enough records)
- But we can generate it!



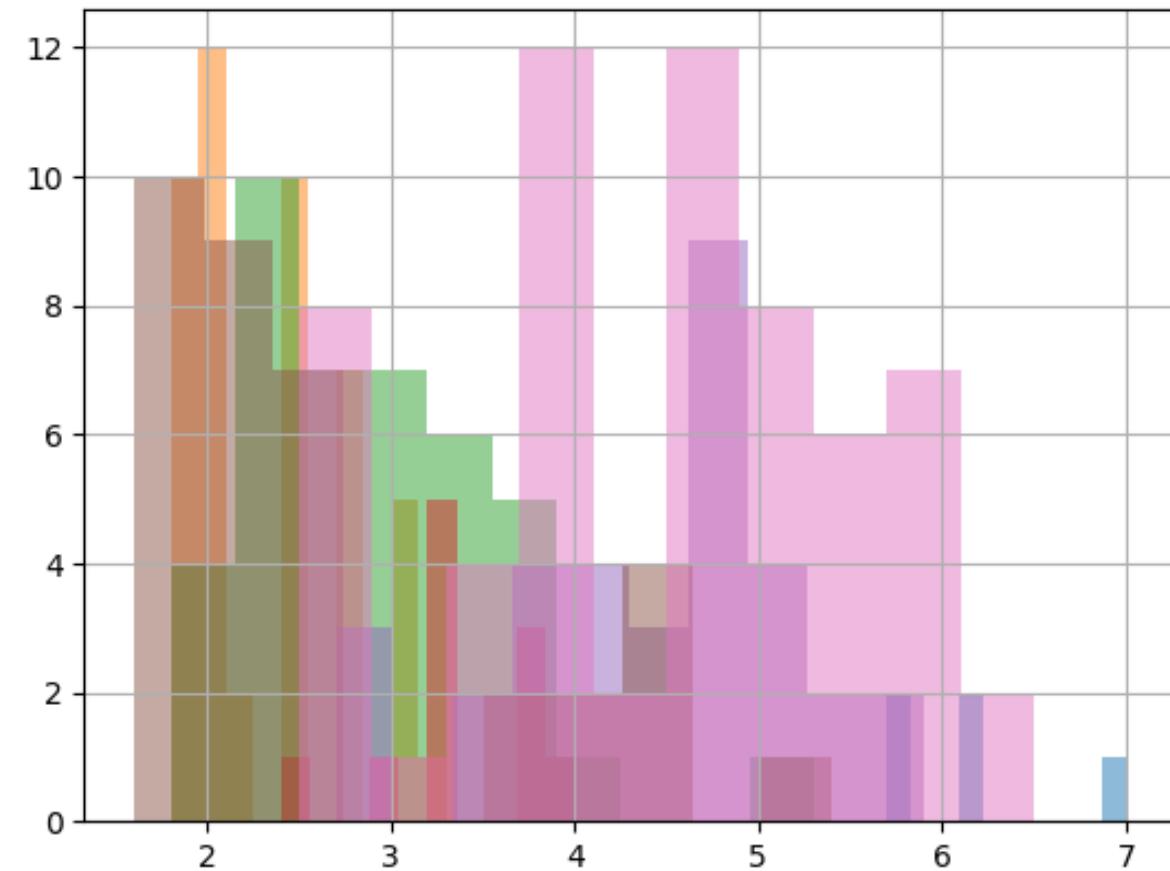
```
import numpy as np  
np.random.seed(1)  
  
data = np.random.normal(loc=0, scale=1, size=200)  
  
figure, axis = plt.subplots(1, 2, figsize=(12,5))  
  
axis[0].hist(data, 45)  
axis[1].boxplot(data)  
plt.show()
```



# Comparing histograms

- The entire hwy-column isn't normal, but maybe when we divide it by class of car?
- Can we show all histograms in one graph?
- Sure!

```
df.groupby('class').displ.hist(alpha=0.5)
```



- But it looks **really** bad...

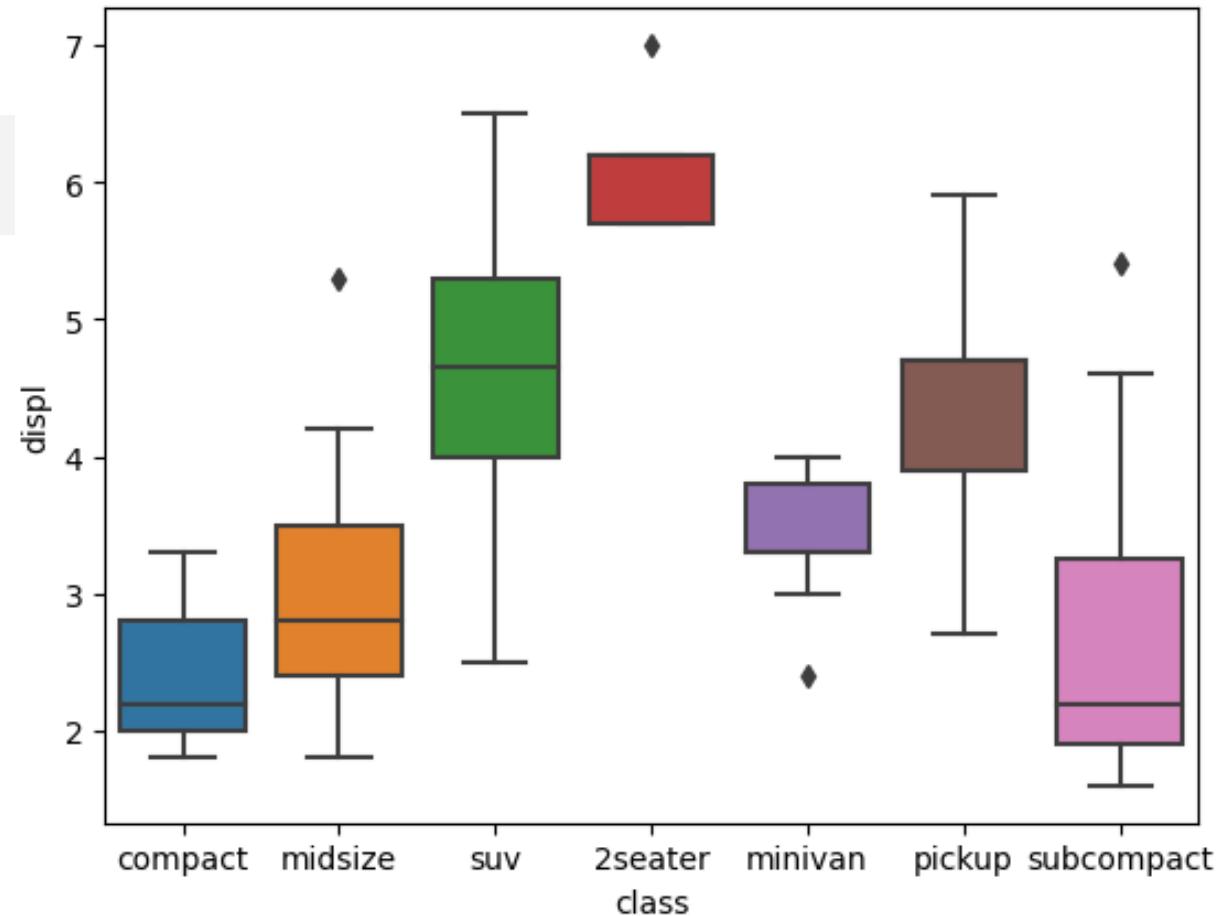


# All hail the boxplot

- Boxplots will fix that mess quickly

```
sns.boxplot(data=df, x='class', y='displ')
```

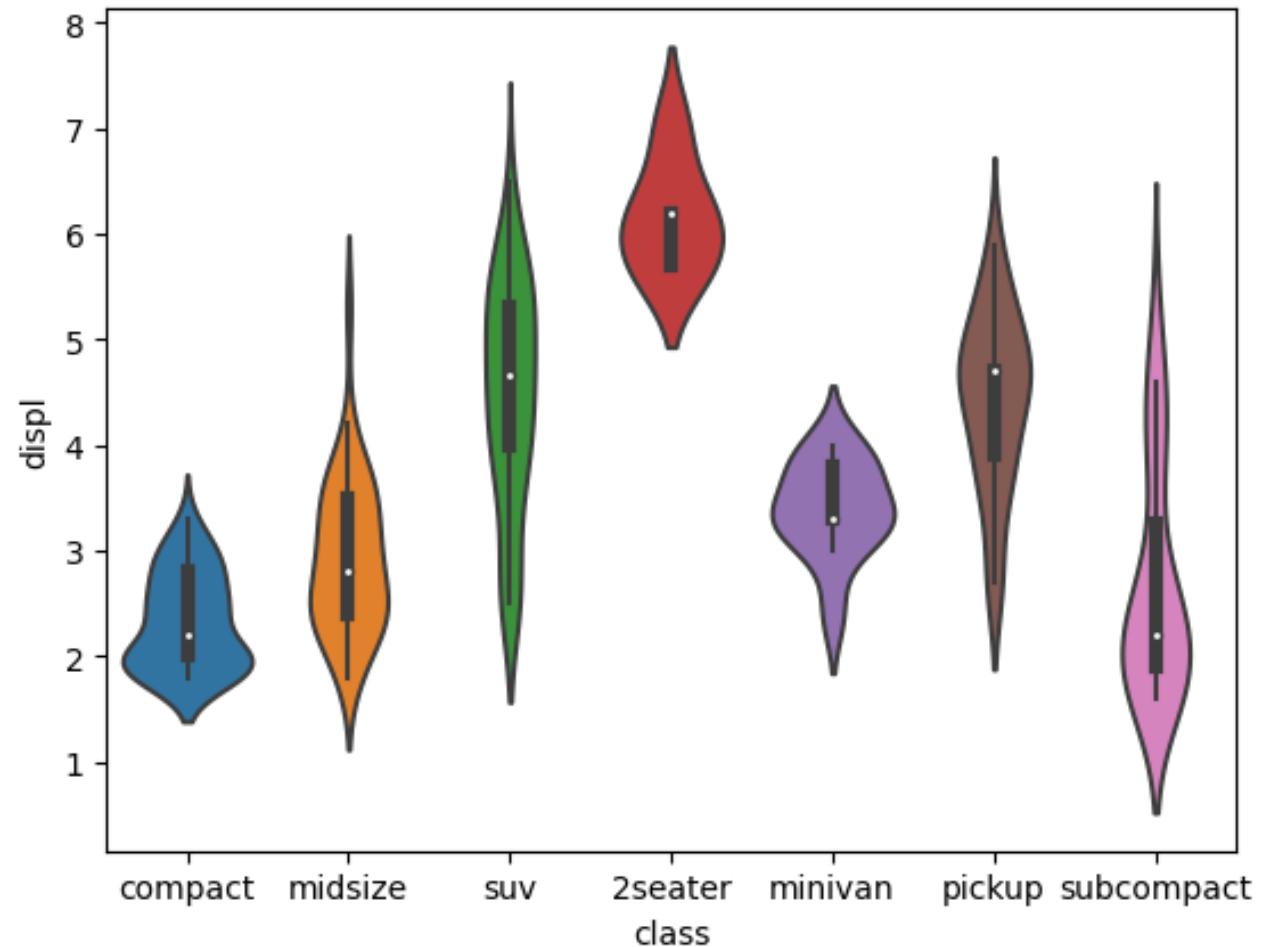
- Remember: boxplots simplify data. Sometimes that is bad, but mostly it helps making data understandable and comparable.
- The graph shown here is clear and tells a story. That is what we need.





# Violins are nice too

- They look better but contain way more information making them harder to understand.
- Violins are good to raise questions (why is purple such an ugly blob?) but **never** to answer them.



```
sns.violinplot(x='class', y='displ', data=df)
```



# Exercises

## 📁 files

- 📄 6.1 - Combine different types of data.ipynb
- 📄 6.2 - Reshape.ipynb
- 📄 6.3 - Crosstab.ipynb
- 📄 6.4 - DatasauRus.ipynb
- 📄 6.5 - Diamonds.ipynb
- 📄 6.6 - Outliers.ipynb