

# Data Science

Lecture 2-1: Data Science Fundamentals (Pipeline)

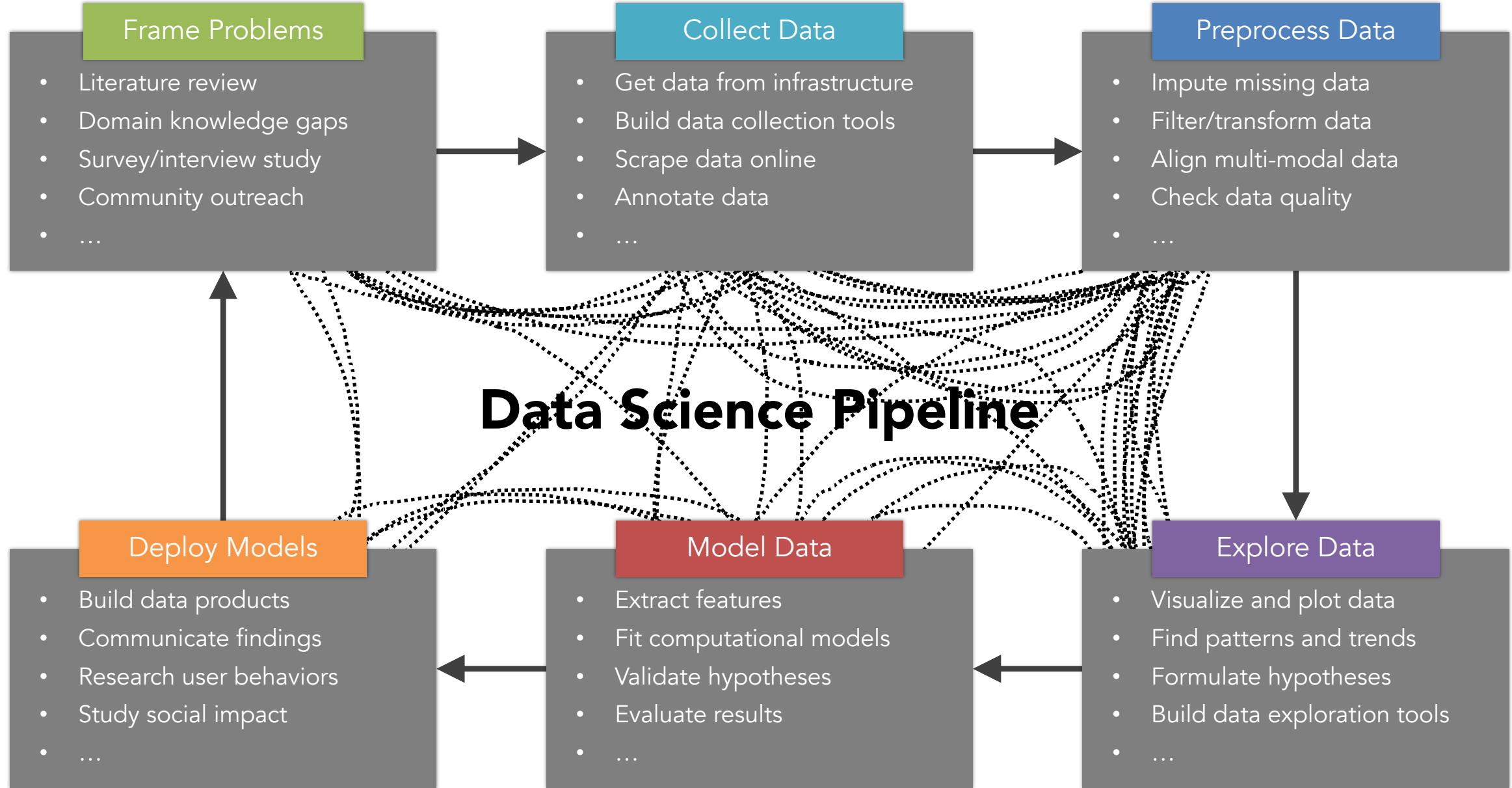


UNIVERSITY  
OF AMSTERDAM

Lecturer: Yen-Chia Hsu

Date: Feb 2025

This lecture shows a typical **data science**  
**pipeline** and recaps **data cleaning** techniques.



What people typically think : →

The reality of the data science pipeline: ..... 3

## Frame Problems

This course uses existing scenarios and cases with well-defined problems. However, in the real world, we need to define and frame the problems first.

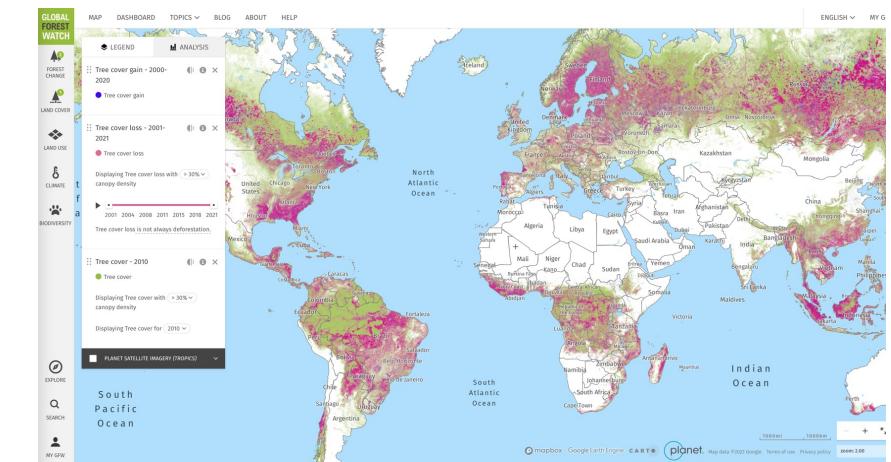
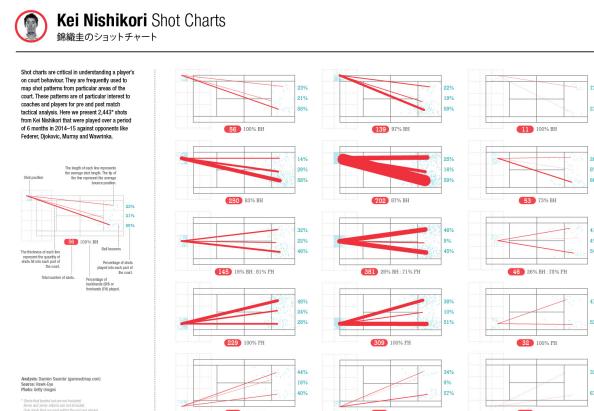
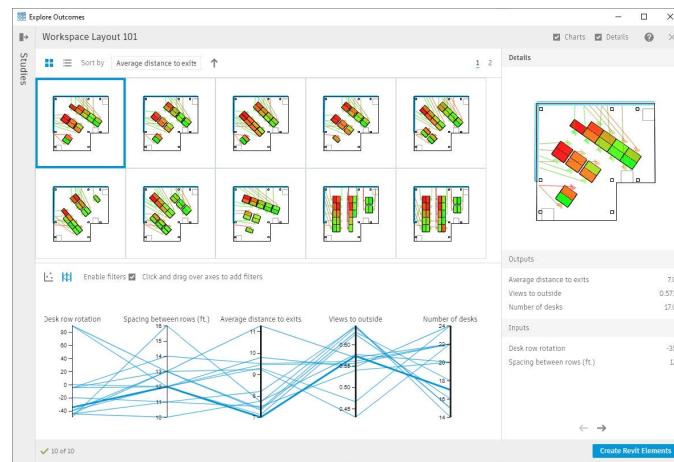
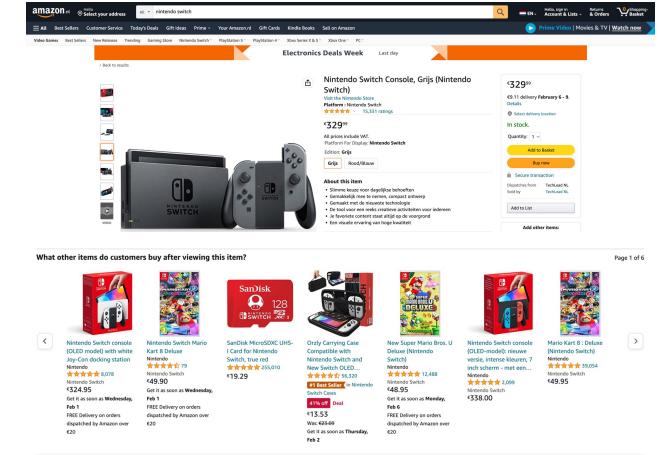
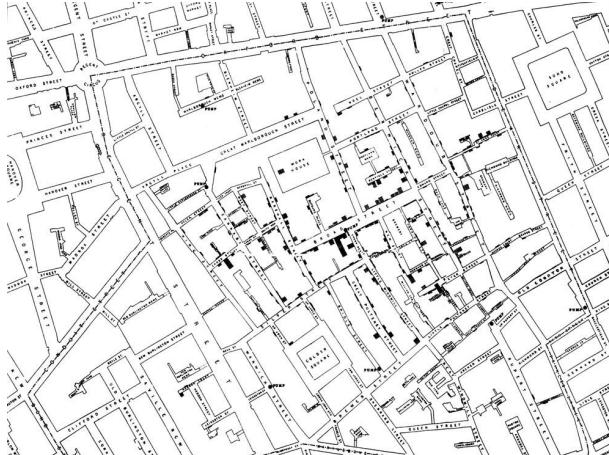


Figure sources are in the slides for the first lecture.

## Collect Data

This course assumes that someone has collected the data for you. In reality, you may need to collect data using sensors, crowdsourcing, mobile apps, etc.

The screenshot shows the 'Populaire kaartlagen' (Popular map layers) section of the GGD Amsterdam Data Portal. It features three thumbnail images: a panoramic view of Amsterdam, a canal scene with buildings, and a street view. Below each image is a title and a brief description:

- Kadastrale perceelsgrenzen**: De Basisregistratie kadaster (BRK) bevat informatie over kadastrale objecten (percelen en appartementsrechten). De kaartlaag kadastrale perceelsgrenzen toont de kadastrale grenzen en de grenzen van de gemeente Amsterdam.
- Meetbouten - Zakkingsnelheid**: De registratie meetbouten bevat de meetgegevens van boutingen in de Amsterdamse panden. Het doel van de meetbouten is het monitoren van de deformatie (zakkingen of stijgingen). Dit wordt inzichtelijk gemaakt op deze kaartlaag.
- Parkeren - Fiscale indelingen**: De kaartlaag parkeren - fiscaal geeft visueel informatie over de fiscale indeling in Amsterdam. Krijg via de kaart inzicht in waar er fiscaal parkeren en waar er niet-fiscaal parkeren van toepassing is.

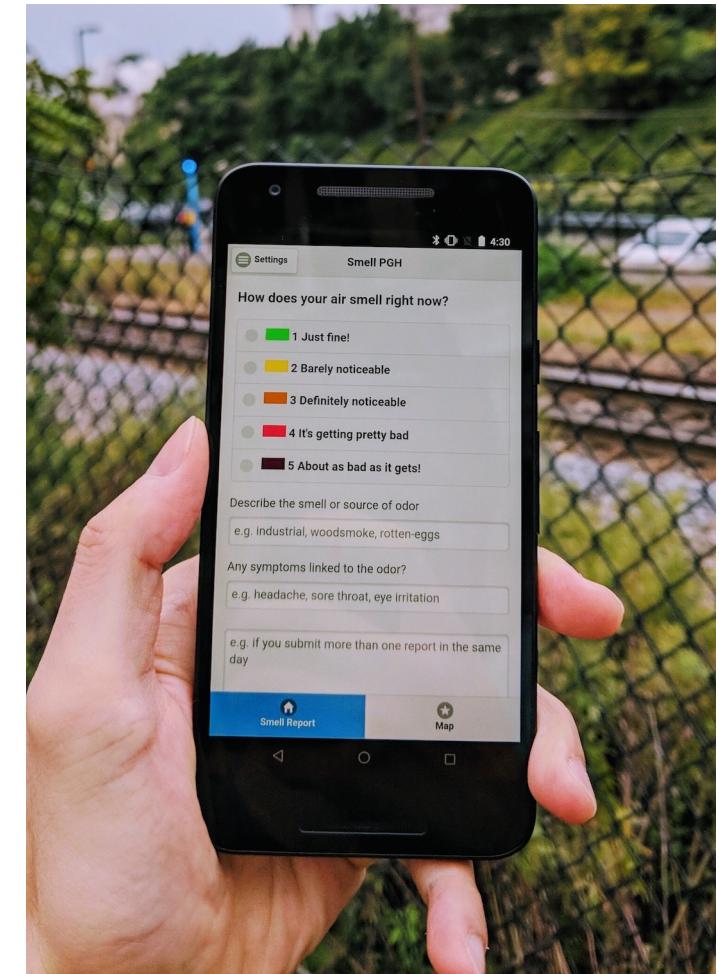
At the bottom, there are links for 'Vragen' (Questions), 'Colofon' (Colophon), and 'Volg ons' (Follow us) with icons for Twitter, Facebook, LinkedIn, and YouTube.

The screenshot shows the Prolific platform interface. The top banner reads 'Set your study live to thousands of reliable participants in minutes.' with a 'Get started' button. Below, a card displays a study titled 'Testing your memory of a crime scene' with the following details:

- £7.50/hour
- 7 mins
- 500 places

Three participant profiles are listed below:

- Chicago, United States**: Time taken: 8 mins
- London, United Kingdom**: Time taken: 10 mins
- Washington, United States**



[GGD Amsterdam Data Portal](#)

[Prolific Tool for Data Annotation](#)

[Mobile App Data Collection](#)

## Collect Data

Hugging Face, Zenodo, Google Dataset Search, government websites, etc.

The Hugging Face dataset search interface shows a list of datasets. Key datasets listed include:

- glue
- openwebtext
- blimp
- imdb
- super\_glue
- red\_caps
- HuggingFaceM4/cm4-synthetic-testing
- wikitext
- textvqa
- squad

[Hugging Face](#)

The Zenodo research shared interface features a "Featured communities" section with a NASA Transform to Open Science badge. It also displays "Recent uploads" and a "Curated by: nasatransformtoopen" section. Recent uploads include:

- Flowminder/FlowKit: 1.18.2
- Trixi.jl

[Zenodo](#)

The Google Dataset Search interface shows results for "sustainability". Key findings include:

- statista: Data sustainability as main consideration in global organizations 2020, by country
- csiro: Australian land-use and sustainability data: 2013 to 2050
- bloomberg: Environmental, Social and Governance Data
- statista: Sustainability reporting rate 2020, by sector

[Google Dataset Search](#)

This course will focus on [pandas](#), which is a very handy Python library for preprocessing structured data. We will cover the following techniques:

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>• Filter unwanted data</li><li>• Aggregate data (e.g., sum)</li><li>• Group data based on a column</li><li>• Sort rows based on a column</li><li>• Concatenate data frames</li><li>• Merge and join data frames</li><li>• Quantize continuous values into bins</li></ul> | <ul style="list-style-type: none"><li>• Scale column values</li><li>• Resample time series data</li><li>• Roll time series data in a window</li><li>• Apply a transformation function</li><li>• Use regular expressions</li><li>• Drop rows or columns</li><li>• Treat missing values</li></ul> |
|--|---|

## Preprocess Data

Filtering can reduce a set of data based on specific criteria. For example, the left table can be reduced to the right table using a population threshold.

D

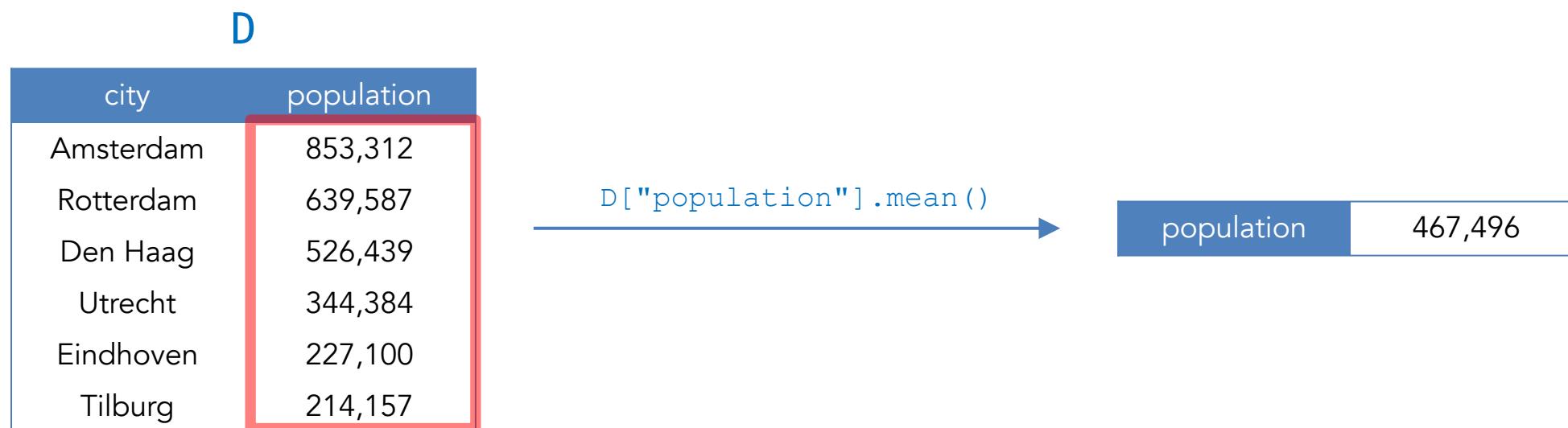
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

D[D["population"]>500000]

city	population
Amsterdam	True
Rotterdam	True
Den Haag	True
Utrecht	False
Eindhoven	False
Tilburg	False

city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439

Aggregation reduces a set of data to a descriptive statistic. For example, the left table is reduced to a single number by computing the mean value.



**Grouping** divides a table into groups by column values, which can be chained with data aggregation to produce descriptive statistics for each group.

city	province	population
Amsterdam	Noord-Holland	853,312
Rotterdam	Zuid-Holland	639,587
Utrecht	Utrecht	344,384
Eindhoven	Noord-Brabant	227,100
Den Haag	Zuid-Holland	526,439
Tilburg	Noord-Brabant	214,157

D

province	population	city
Noord-Holland	853,312	Amsterdam
Zuid-Holland	639,587 526,439	Rotterdam Den Haag
Utrecht	344,384	Utrecht
Noord-Brabant	227,100 214,157	Eindhoven Tilburg

```
D.groupby("province").sum()
```



province	population
Noord-Holland	853,312
Zuid-Holland	1,166,026
Utrecht	344,384
Noord-Brabant	441,257

Sorting rearranges data based on values in a column, which can be useful for inspection. For example, the right table is sorted by population.

D

city	population
Eindhoven	227,100
Den Haag	526,439
Tilburg	214,157
Rotterdam	639,587
Amsterdam	853,312
Utrecht	344,384

`D.sort_values(by=["population"])`

city	population
Tilburg	214,157
Eindhoven	227,100
Utrecht	344,384
Den Haag	526,439
Rotterdam	639,587
Amsterdam	853,312

Concatenation combines multiple datasets that have the same variables. For example, the two left tables can be concatenated into the right table.

A

city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439

B

city	population
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

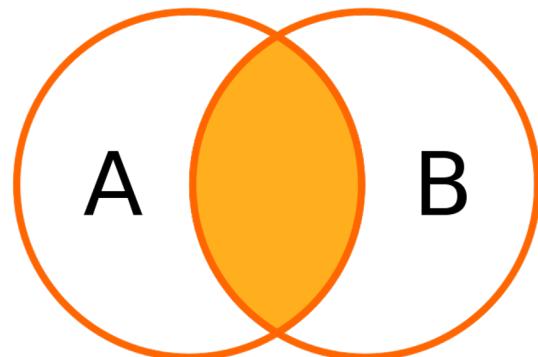
`pandas.concat([A, B])`



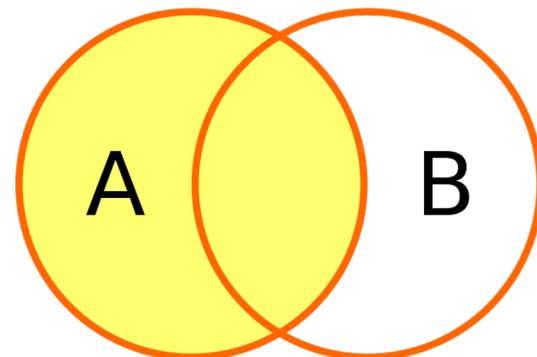
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

Merging and joining is a common method (in relational databases) to merge multiple data tables which have overlapping set of instances.

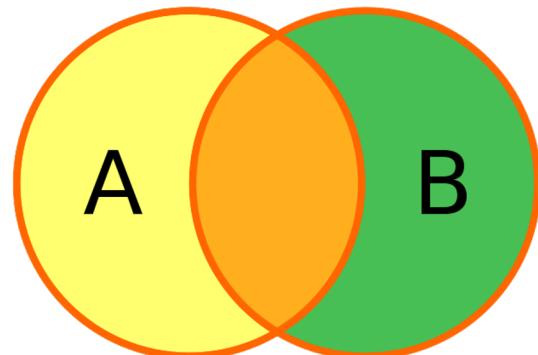
- Inner join



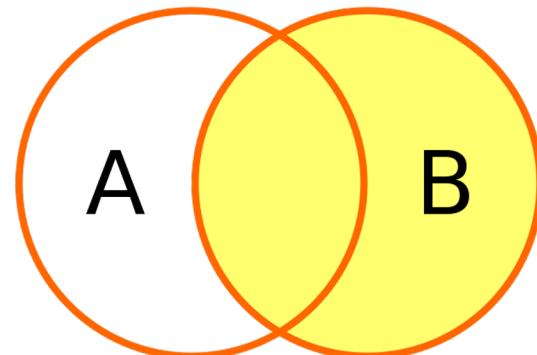
- Left (outer) join



- Outer join



- Right (outer) join



A

city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

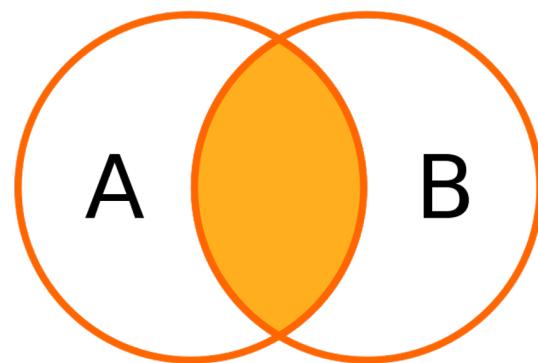
B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="inner", on="city")`

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8



- Inner join

A

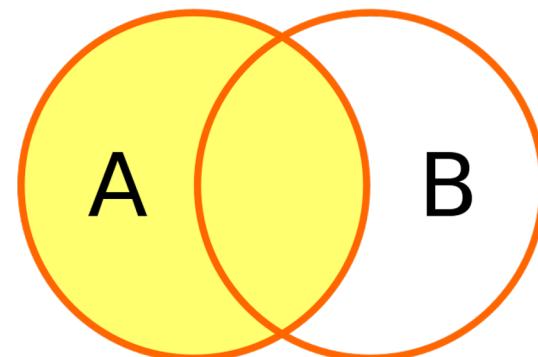
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="left", on="city")`



- Left join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	NaN

A

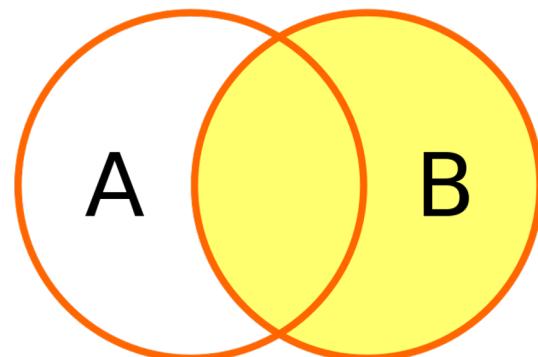
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="right", on="city")`



- Right join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Zwolle	NaN	40.9

A

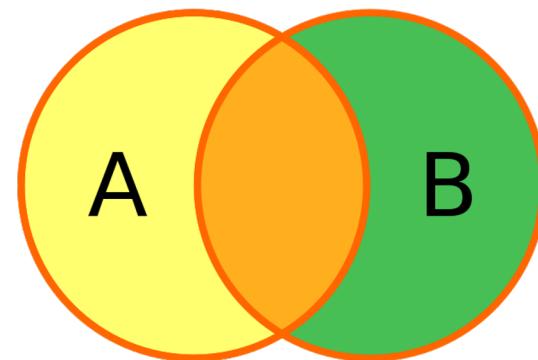
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="outer", on="city")`



- Outer join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	NaN
Zwolle	NaN	40.9

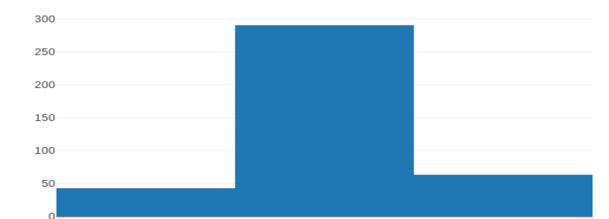
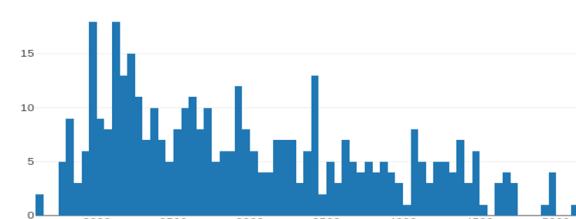
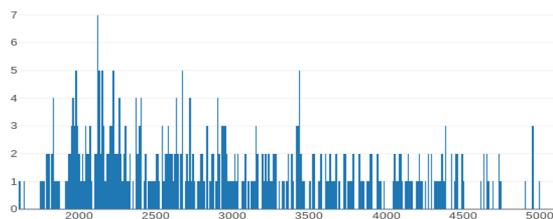
**Quantization** transforms a continuous set of values (e.g., integers) into a discrete set (e.g., categories). For example, age is quantized to age range.

D

name	age
Jantje	8
Piet	16
Maria	22
Renske	34
Donald	65

```
bin = [0,20,50,200]
L = ["1-20","21-50","51+"]
pandas.cut(D["age"], bin, labels=L)
```

name	age
Jantje	1-20
Piet	1-20
Maria	21-50
Renske	21-50
Donald	51+



Scaling transforms variables to have another distribution, which puts variables at the same scale and makes the data work better on many models.

	D
population	air_quality
853,312	42.4
639,587	40.9
526,439	41.1
344,384	41.4
227,100	43.8
214,157	39.1

- Z-score scaling (representing how many standard deviations from the mean)

$$(D - D.\text{mean}()) / D.\text{std}()$$

- Min-max scaling (making the value range between 0 and 1)

$$(D - D.\text{min}()) / (D.\text{max}() - D.\text{min}())$$

population	air_quality
1.5273	0.6039
0.6812	-0.3496
0.2333	-0.2225
-0.4874	-0.0318
-0.9516	1.4938
-1.0029	-1.4938

population	air_quality
1	0.7021
0.6656	0.3830
0.4886	0.4255
0.2037	0.4894
0.0203	1
0	0

## Preprocess Data

You can **resample** time series data (i.e., the data with time stamps) to a different frequency (e.g., hourly) using different aggregation methods (e.g., mean).

D

timestamp	v1
2016-10-31 07:30:00	52.60
2016-10-31 08:30:00	48.30
2016-10-31 08:53:20	44.20
2016-10-31 09:30:00	31.10

`D.resample("60Min", label="right").mean()`



timestamp	v1
2016-10-31 08:00:00	52.60
2016-10-31 09:00:00	46.25
2016-10-31 10:00:00	31.10

## Preprocess Data

You can use the `rolling` window operation to transform time series data using different aggregation methods (e.g., sum).

D

timestamp	v1
2016-10-31 08:00:00	52.60
2016-10-31 09:00:00	46.25
2016-10-31 10:00:00	31.10
2016-10-31 11:00:00	12.21
2016-10-31 12:00:00	28.64

`D["v2"] = D["v1"].rolling(window=3).sum()`

timestamp	v2
2016-10-31 08:00:00	NaN
2016-10-31 09:00:00	NaN
2016-10-31 10:00:00	129.95
2016-10-31 11:00:00	89.56
2016-10-31 12:00:00	71.95

You can apply a **transformation** to rows or columns in the data frame.

D

wind_mph
3.6
NaN
5.1

```
def f(x):
    if pd.isna(x): return None
    else: return x<5
D["is_calm"] = D["wind_mph"].apply(f)
```

wind_mph	is_calm
3.6	True
NaN	None
5.1	False

D

wind_deg
343
351
359
5
41
25
:



Very slow if you have a lot of rows!

```
def f(x):
    return numpy.sin(numpy.deg2rad(x))
D["wind_sine"] = D["wind_deg"].apply(f)
```

```
D["wind_sine"] = np.sin(np.deg2rad(D["wind_deg"]))
```



Better to transform the entire column directly!

wind_deg	wind_sine
343	-0.292372
351	-0.156434
359	-0.017452
5	0.087156
41	0.656059
25	0.422618
:	:

## Preprocess Data

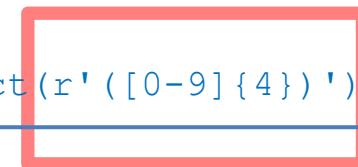
To extract data from text or match text patterns, you can use **regular expression**, which is a language to specify search patterns.

D

venue
WACV_2023
WACV
2023NeurIPS
CVPR2022

```
D["year"] = D["venue"].str.extract(r'([0-9]{4})')
```

This means matching pattern with 4 digits



venue	year
WACV_2023	2023
WACV	NaN
2023NeurIPS	2023
CVPR2022	2022

## Preprocess Data

We can **drop** data that we do not need, such as duplicate data records or those that are irrelevant to our research question.

city	population	year
Amsterdam	853,312	2018
Rotterdam	639,587	2018
Den Haag	526,439	2018

`pandas.drop(columns=["year"])`

city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439

	city	population	year
0	Amsterdam	853,312	2018
1	Rotterdam	639,587	2018
2	Den Haag	526,439	2018
3	Utrecht	344,384	2018
4	Eindhoven	227,100	2018
5	Amsterdam	862,965	2019
6	Utrecht	344,384	2018

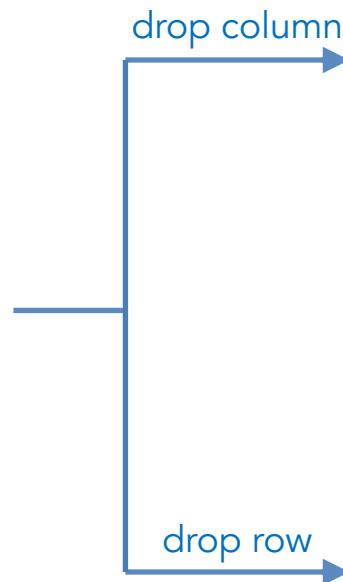
`pandas.drop([5, 6])`

	city	population	year
0	Amsterdam	853,312	2018
1	Rotterdam	639,587	2018
2	Den Haag	526,439	2018
3	Utrecht	344,384	2018
4	Eindhoven	227,100	2018

## Preprocess Data

We can either drop the rows (i.e., the records/observations) or the columns (i.e., the variables/attributes) that contain the missing values.

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	



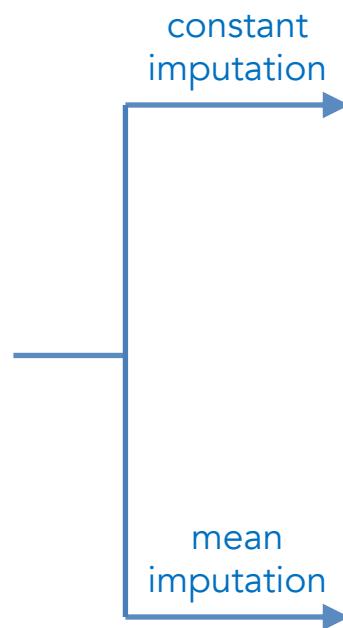
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8

## Preprocess Data

We can replace the missing values (i.e., imputation) with a constant, mean, median, or the most frequent value along the same column.

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	



city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	-1

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	41.92

## Preprocess Data

We can model missing values, where  $y$  is the variable/column that has the missing values,  $X$  means other variables, and  $F$  is a regression function.

city	population ( $X$ )	air_quality ( $y$ )
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	

$$\textcolor{blue}{y} = F(\textcolor{red}{X}) \rightarrow$$

city	population ( $X$ )	air_quality ( $y$ )
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	42.46

Different missing data may require different data cleaning methods.

Missing Not At Random is a big problem and cannot be solved simply with imputation.

## MCAR

Missing Completely At Random:

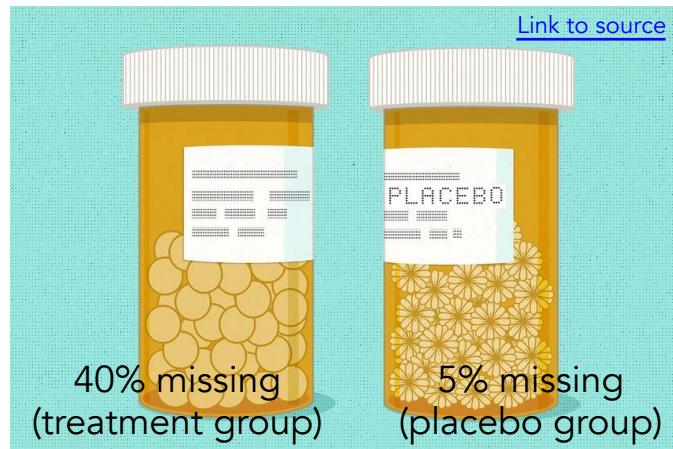
- Missing data is a completely random subset (no relations) of the entire dataset.



## MAR

Missing at Random:

- Missing data is only related to variables other than the one having missing data.



## MNAR

Missing Not At Random:

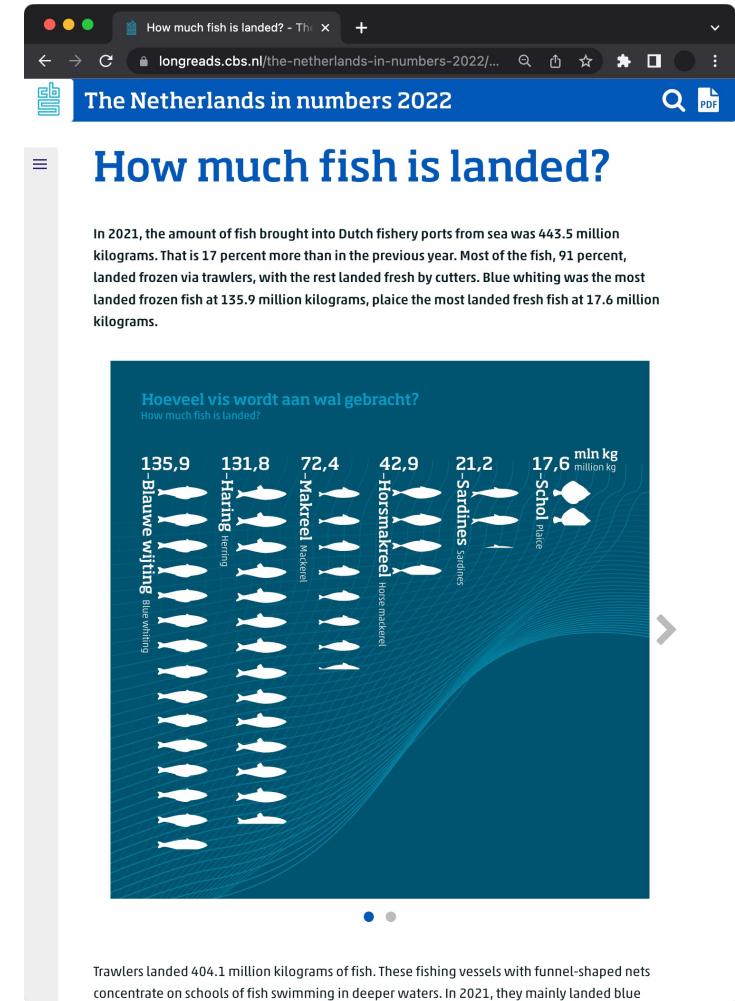
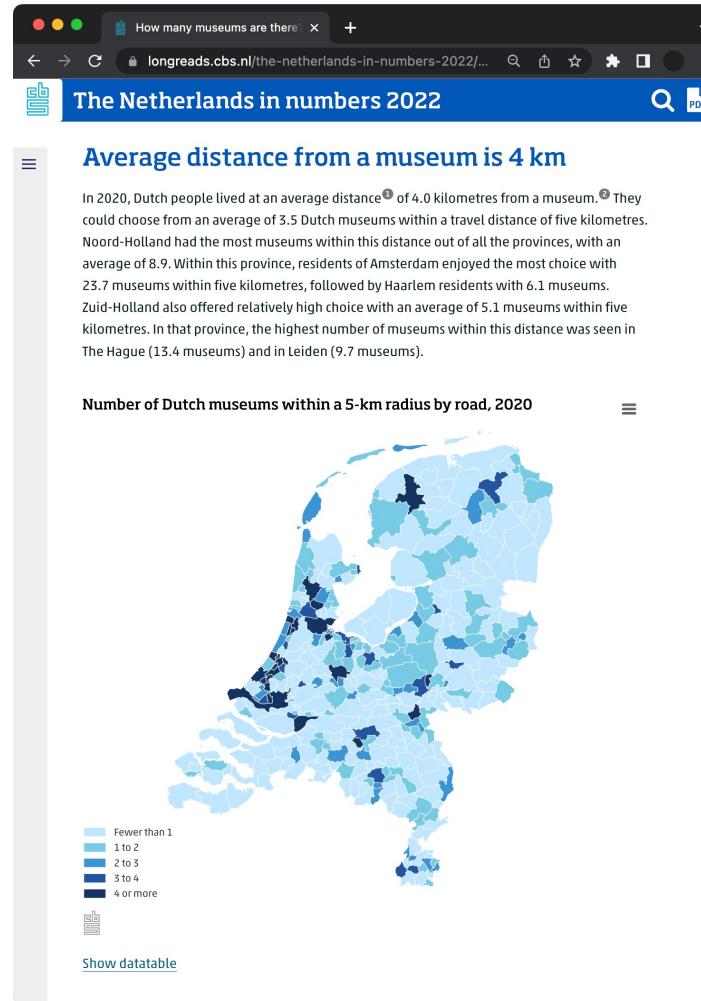
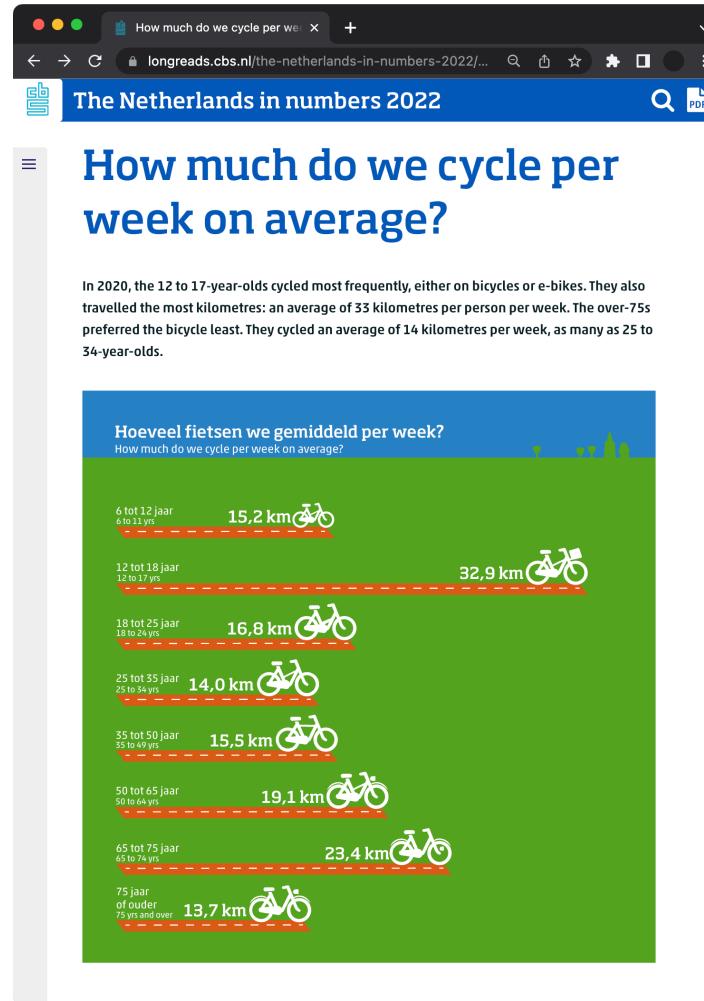
- Missing data is related to the variable that has the missing data. (e.g., sensitive questions)

A screenshot of a survey interface. It shows a question: "Do you have any history of mental illness in your family? If yes, who in your family?" Below the question are two radio buttons: one for "No" and one for "Other". There is also a text input field for "Other" responses.

You really need to practice coding a lot to  
know and internalize how these things work!

- [Pandas exercises on GitHub](#)
- [Pandas exercises on Kaggle](#)
- [Pandas exercises on W3Schools](#)
- [Pandas exercises by UC Berkeley School of Information](#)
- [Pandas exercises on GeeksforGeeks](#)
- [Pandas exercises on w3resource](#)

# Information visualization is a good way for both experts and laypeople to explore data and gain insights.



## Explore Data

You can use the Python seaborn library (based on matplotlib) to quickly plot and explore structured data.

The screenshot shows the official Seaborn website at [seaborn.pydata.org/index.html](https://seaborn.pydata.org/index.html). The page features a header with the Seaborn logo, navigation links for Installing, Gallery, Tutorial, API, Releases, Citing, and FAQ, and social media icons for search, GitHub, YouTube, and Twitter. Below the header is a section titled "seaborn: statistical data visualization" displaying six examples of Seaborn plots: a joint plot with marginal distributions, a density plot with multiple layers, a scatter plot with a regression line, contour plots for two years (1955 and 1958), a box plot with violin plots, and a scatter plot with a linear regression model. The main content area includes a brief introduction, installation instructions, and links to introductory notes, the paper, example gallery, tutorials, API reference, GitHub repository, and Stack Overflow support. On the right, there are "Contents" and "Features" sections with links to the respective API and tutorial pages for various plot types and features.

seaborn: statistical data visualization

Seaborn is a Python data visualization library based on [matplotlib](#). It provides a high-level interface for drawing attractive and informative statistical graphics.

For a brief introduction to the ideas behind the library, you can read the [introductory notes](#) or the [paper](#). Visit the [installation](#) page to see how you can download the package and get started with it. You can browse the [example gallery](#) to see some of the things that you can do with seaborn, and then check out the [tutorials](#) or [API reference](#) to find out how.

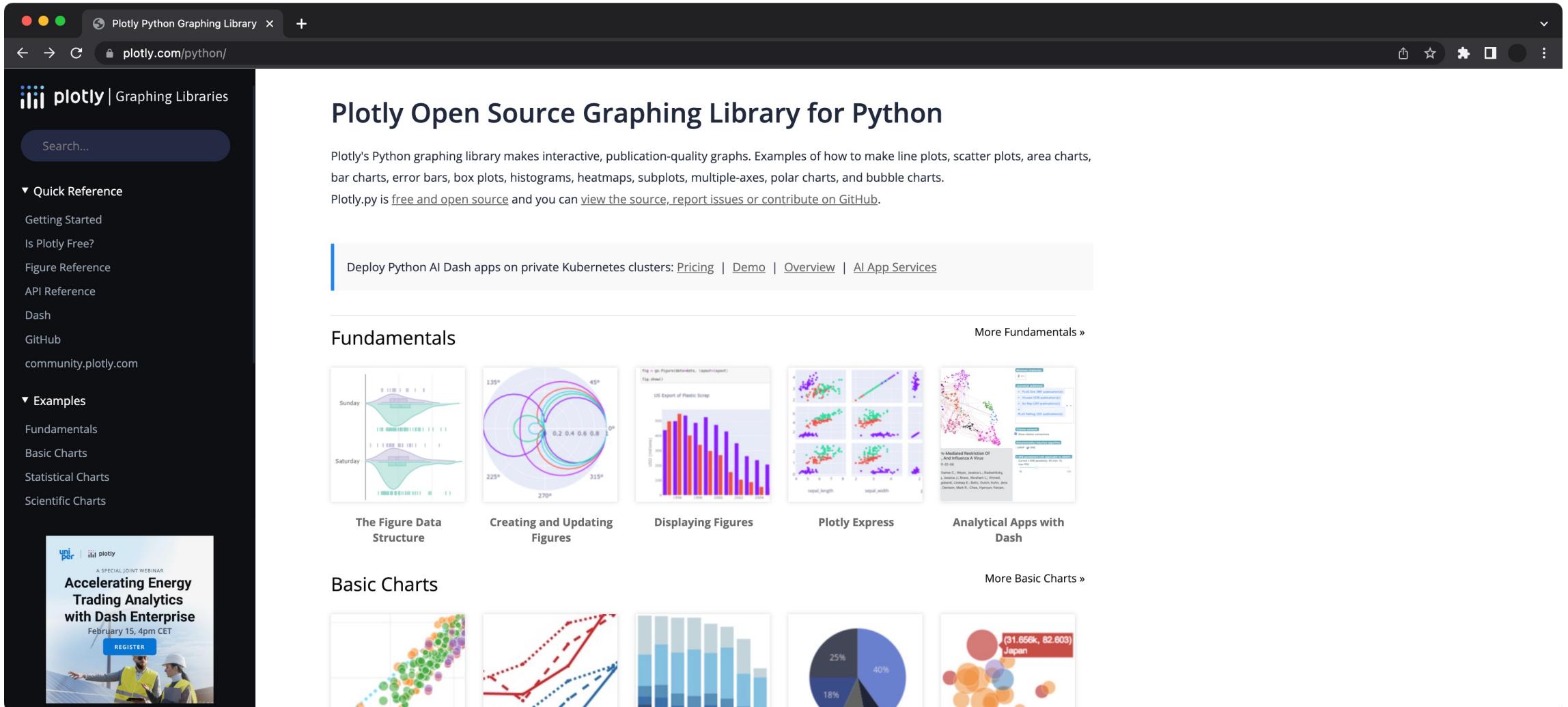
To see the code or report a bug, please visit the [GitHub repository](#). General support questions are most at home on [Stack Overflow](#), which has a dedicated channel for seaborn.

**Contents**

- [Installing](#)
- [Gallery](#)
- [Tutorial](#)
- [API](#)
- [Releases](#)
- [Citing](#)
- [FAQ](#)

**Features**

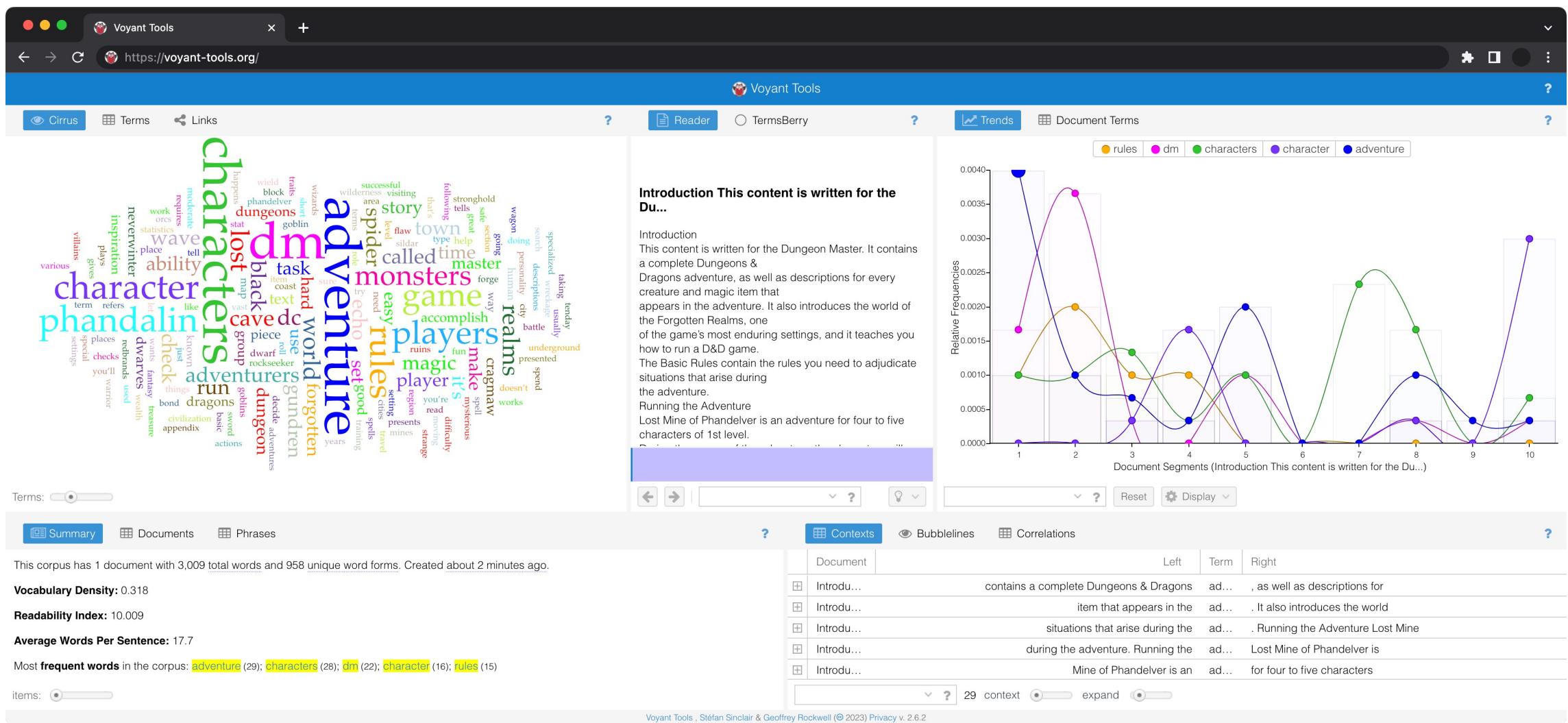
- **New** Objects: [API](#) | [Tutorial](#)
- Relational plots: [API](#) | [Tutorial](#)
- Distribution plots: [API](#) | [Tutorial](#)
- Categorical plots: [API](#) | [Tutorial](#)
- Regression plots: [API](#) | [Tutorial](#)
- Multi-plot grids: [API](#) | [Tutorial](#)
- Figure theming: [API](#) | [Tutorial](#)
- Color palettes: [API](#) | [Tutorial](#)



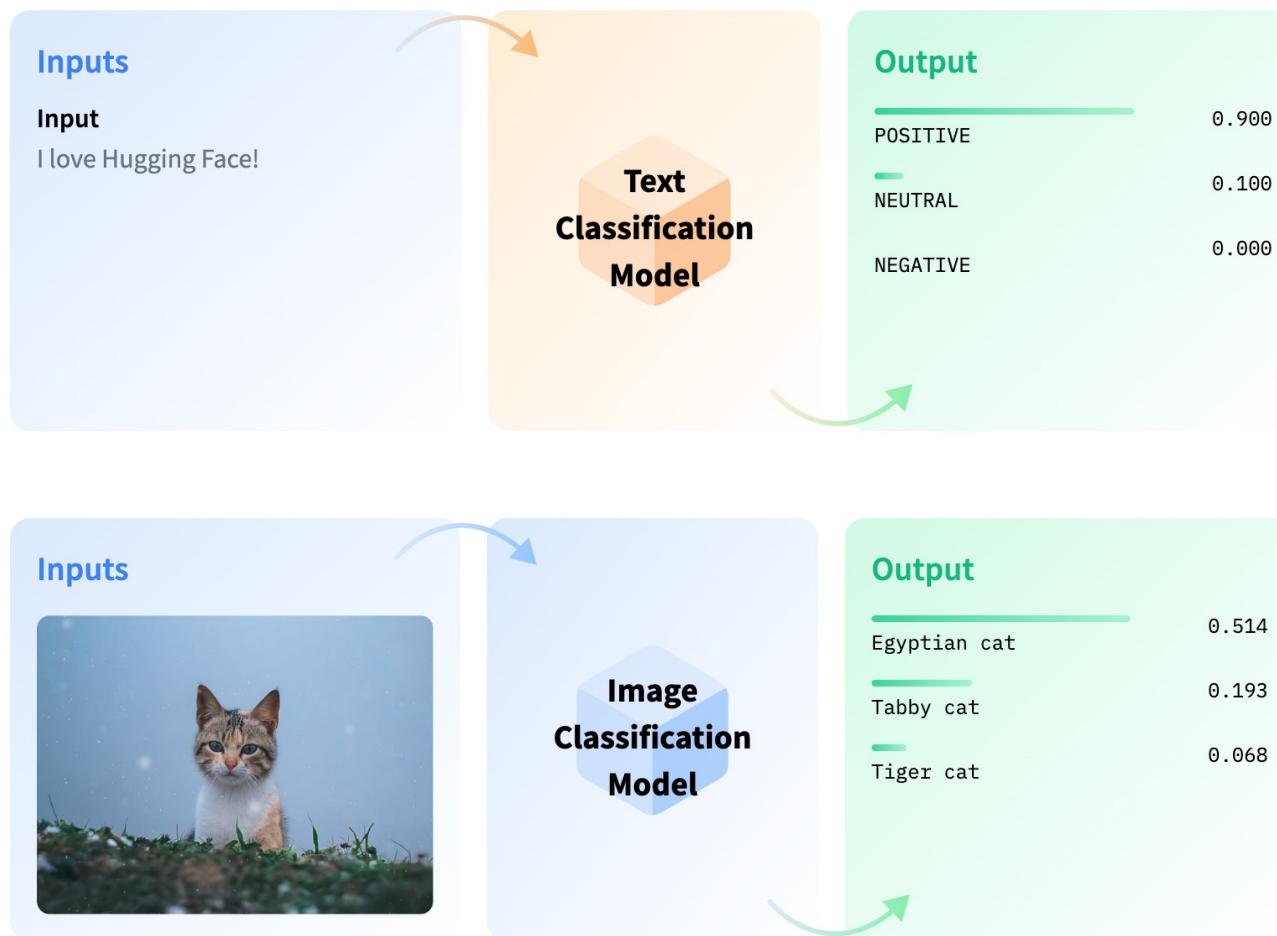
The screenshot shows the official Plotly Python Graphing Library website at <https://plotly.com/python/>. The page has a dark-themed header with the Plotly logo and navigation links for "Search...", "Quick Reference", "Examples", and "Fundamentals". A sidebar on the left lists "Getting Started", "Is Plotly Free?", "Figure Reference", "API Reference", "Dash", "GitHub", and "community.plotly.com". A promotional banner for a joint webinar with UniPer is visible. The main content area features sections for "Fundamentals" (with examples like a sunburst chart, a Venn diagram, a histogram, and a scatter plot grid), "Basic Charts" (with examples like a bubble chart, a line chart with multiple series, a bar chart, and a pie chart), and "More Fundamentals" and "More Basic Charts" sections. A callout box at the top right promotes deploying AI Dash apps on Kubernetes clusters with links to Pricing, Demo, Overview, and AI App Services.

# Explore Data

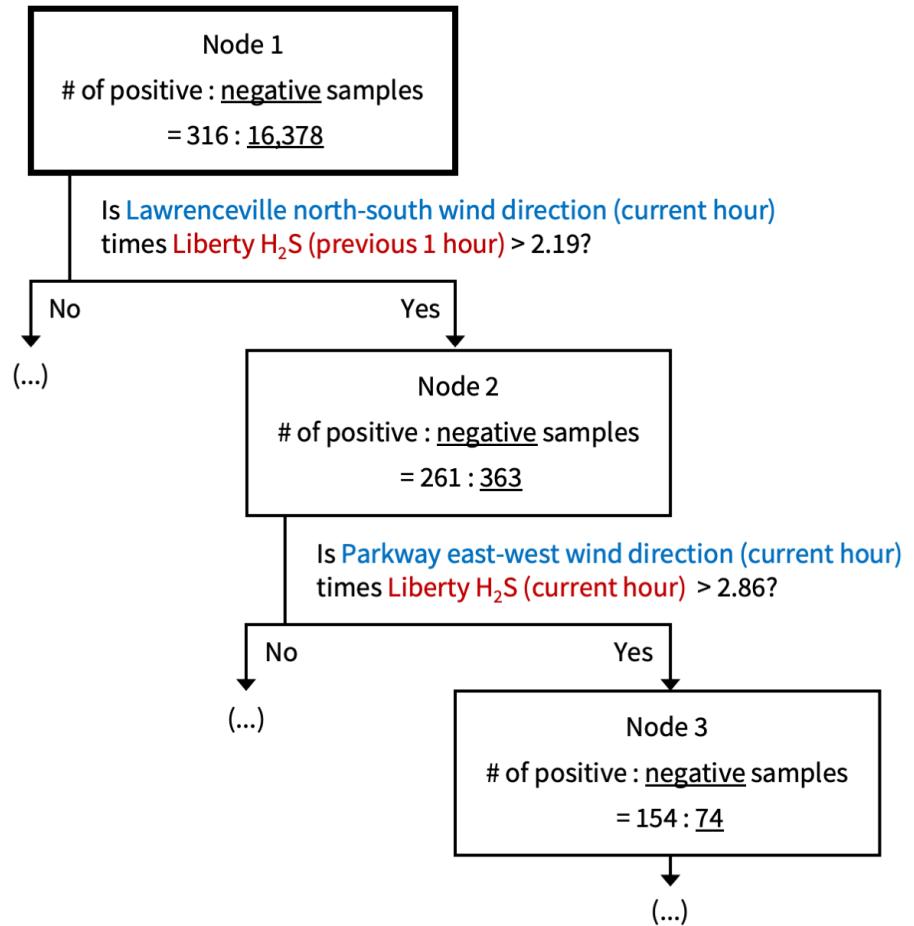
You can use the Voyant Tools to explore text data.



Model Data This course will teach you techniques for modeling structured, text, and image data through three modules from a practical point of view.

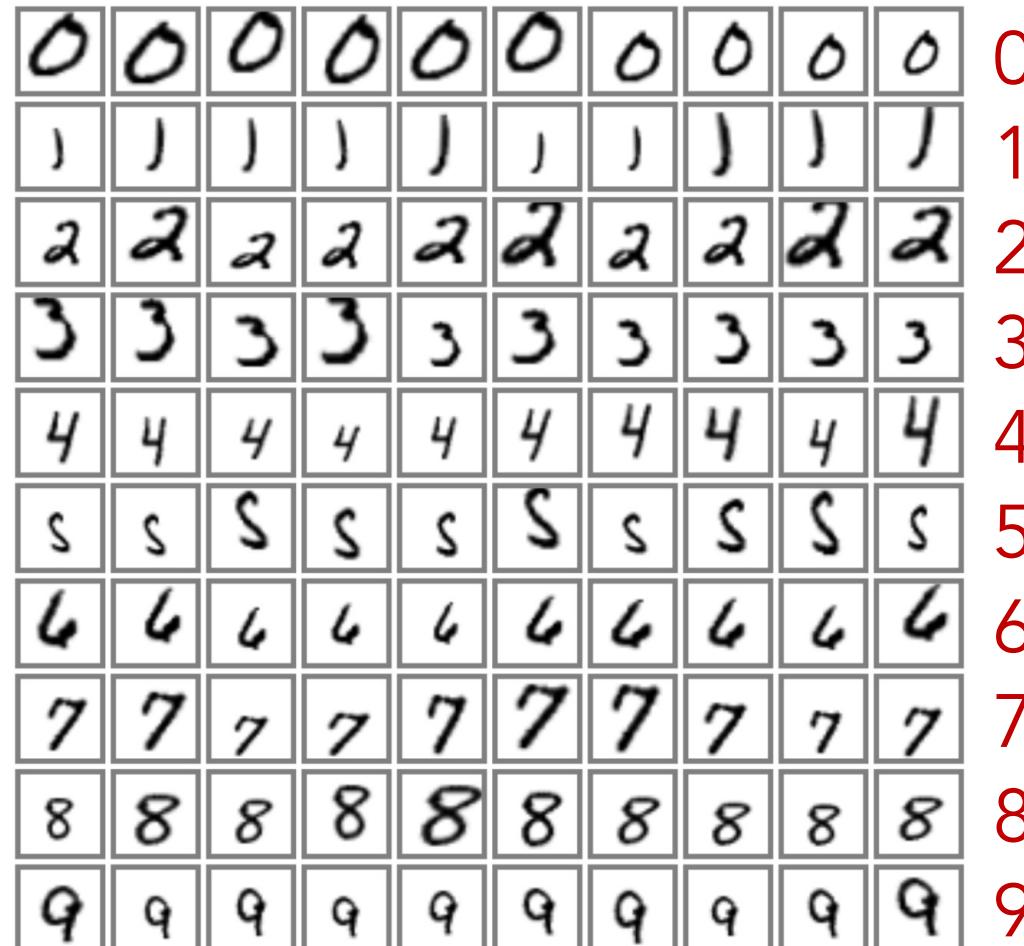


Source: <https://huggingface.co/tasks>



Source: <https://smellpgh.org/>

One example of image classification is **optical character recognition**, such as recognizing digits from hand-written images.



A more complicated image classification task is **fine-grained categorization**, such as categorizing the types of birds.



Barred Owl



American Robin



American Crow



Rufous Hummingbird



Rock Pigeon



Canada Goose

One example of text classification is **sentiment analysis**, such as identifying emotions from movie reviews.



## STAR WARS: THE LAST JEDI REVIEWS

All Critics    **Top Critics**    All Audience

◀ Page 1 of 4 ▶

<b>Matthew Rozsa</b> Salon.com		"Star Wars" is not "Breaking Bad," and the same narrative tricks that worked for the latter feel jarringly out of place in the former. <a href="#">Full Review</a>   Original Score: 2/4	
<b>Leah Pickett</b> Chicago Reader		What's most interesting to me about The Last Jedi is Luke's return as the mentor rather than the student, grappling with his failure in this new role, and later aspiring to be the wise and patient teacher. <a href="#">Full Review</a>   Original Score: 3/4	
<b>Jake Wilson</b> The Age (Australia)		While The Last Jedi may not receive top marks for originality, the eighth official entry in the Star Wars saga is still one of the most entertaining blockbusters of the year... <a href="#">Full Review</a>   Original Score: 3.5/5	
<b>Peter Rainer</b> Christian Science Monitor		Fanatics will love it; for the rest of us, it's a tolerably good time. <a href="#">Full Review</a>   Original Score: B	
<b>Matthew Lickona</b> San Diego Reader		The devoted will no doubt be delighted; for the rest, a resigned acceptance may be the safest path to enjoyment. <a href="#">Full Review</a>   Original Score: 2/5	

A more complex text classification task is annotating paragraphs, such as **categorizing the research aspect** for each fragment in the paper abstract.

For successful infection, viruses must recognize their respective host cells. A common mechanism of host recognition by viruses is to utilize a portion of the host cell as a receptor. Bacteriophage Sf6, which infects *Shigella flexneri*, uses lipopolysaccharide as a primary receptor and then requires interaction with a secondary receptor, a role that can be fulfilled by either outer membrane proteins (Omp) A or C. Our previous work showed that specific residues in the loops of OmpA mediate Sf6 infection. To better understand Sf6 interactions with OmpA loop variants, we determined the kinetics of these interactions through the use of biolayer interferometry, an optical biosensing technique that yields data similar to surface plasmon resonance. Here, we successfully tethered whole Sf6 virions, determined the binding constant of Sf6 to OmpA to be 36 nM. Additionally, we showed that Sf6 bound to five variant OmpAs and the resulting kinetic parameters varied only slightly. Based on these data, we propose a model in which Sf6: Omp receptor recognition is not solely based on kinetics, but likely also on the ability of an Omp to induce a conformational change that results in productive infection.  
All rights reserved. No reuse allowed without permission.

Background

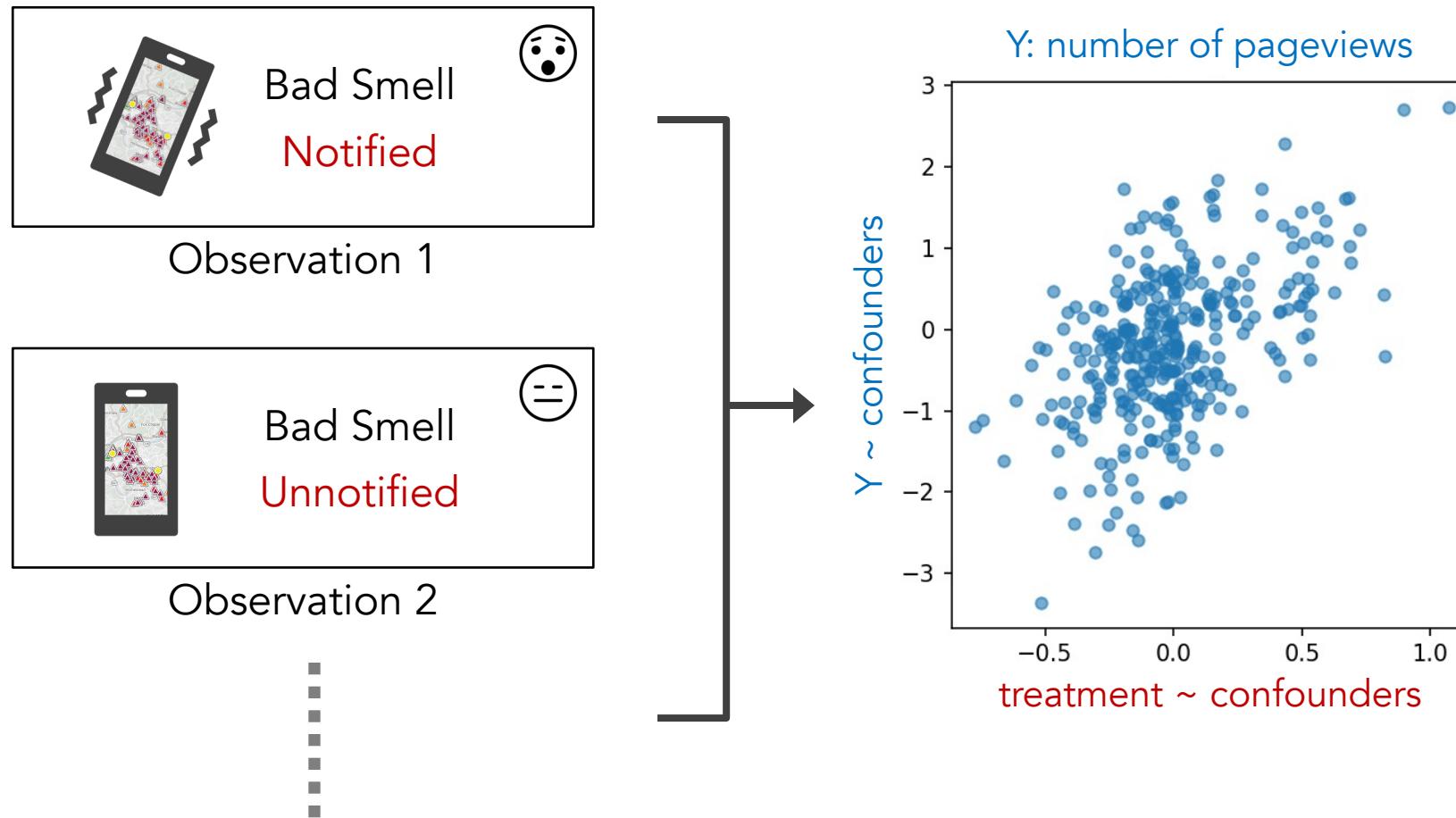
Purpose

Method

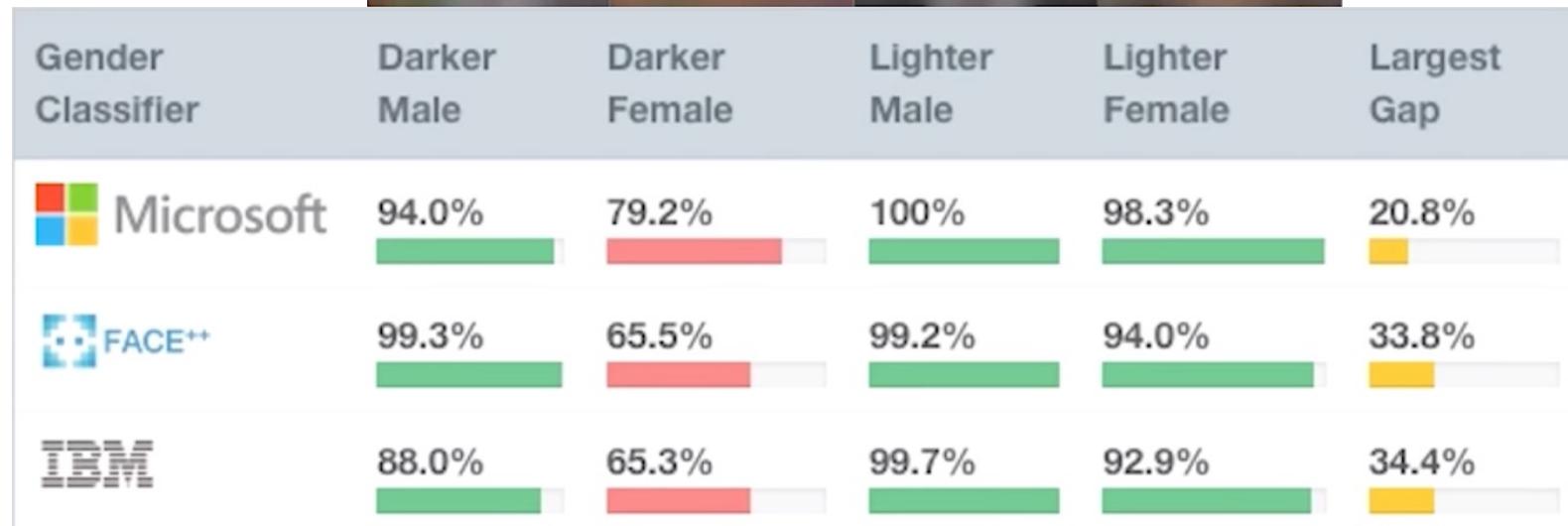
Finding

Other

Deploying models in the wild can enable further quantitative or qualitative research with insights, such as the push notification study in Smell Pittsburgh.



Another example is to study the behaviors of deployed systems to understand its social impact. For example, face recognition systems for recognizing gender did **worst on darker-skin female images**.



# Take-Away Messages

- The data science pipeline is not always linear. Be flexible and open-minded!
- Be aware of the step of framing problems. This course assumes that the problems are defined.
- Collecting data requires well-designed software/hardware infrastructure.
- Being familiar with pandas can speed up the data preprocessing step.
- Different types of missing data require different treatments.
- Besides using descriptive statistics, it is also a good idea to visualize and explore data.
- Different types of data need different modeling techniques. There is no “one solution for all”.
- It is important to study user behaviors and investigate the social impact of deployed models.



# Questions?