

Data Science

Lecture 2-1: Data Science Fundamentals (Pipeline)

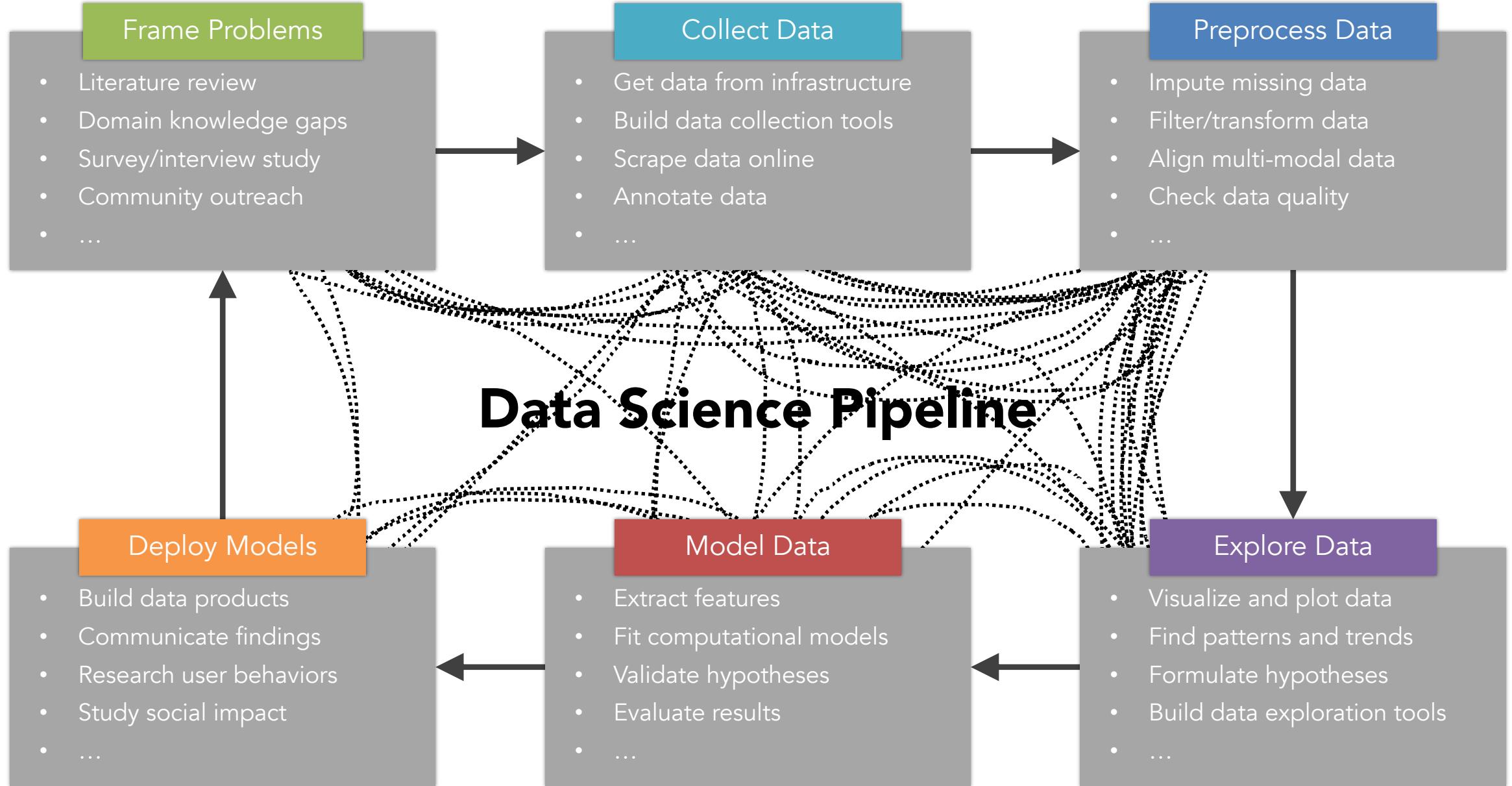


UNIVERSITY
OF AMSTERDAM

Lecturer: Yen-Chia Hsu

Date: Feb 2023

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



Frame Problems

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

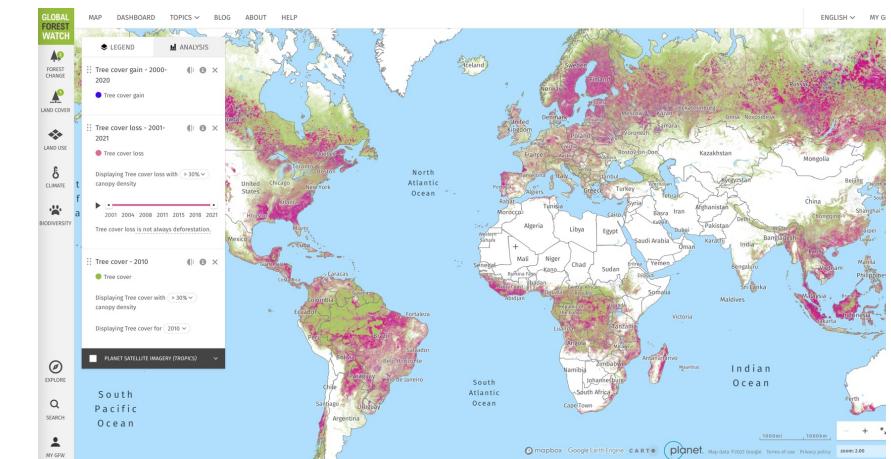
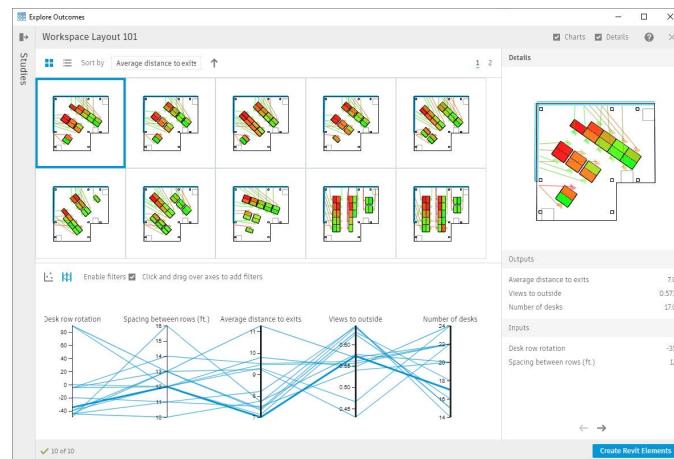
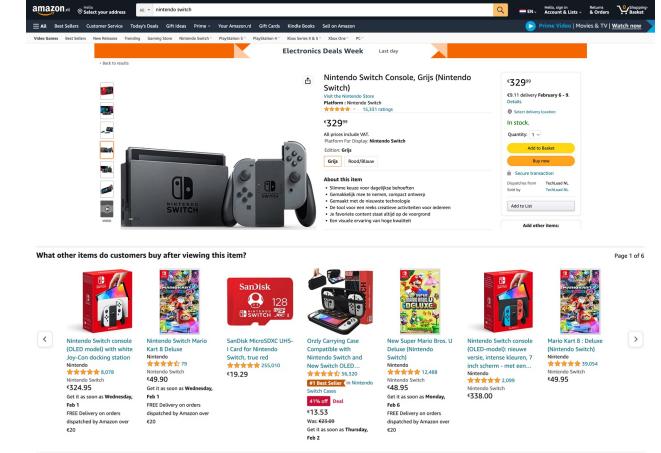
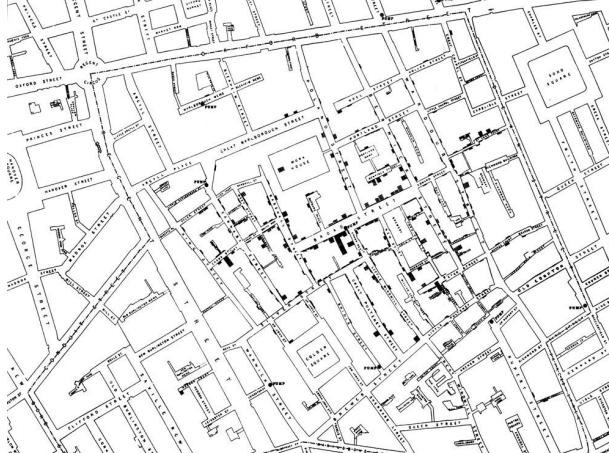


Figure sources are in the slides for the first lecture.

Collect Data

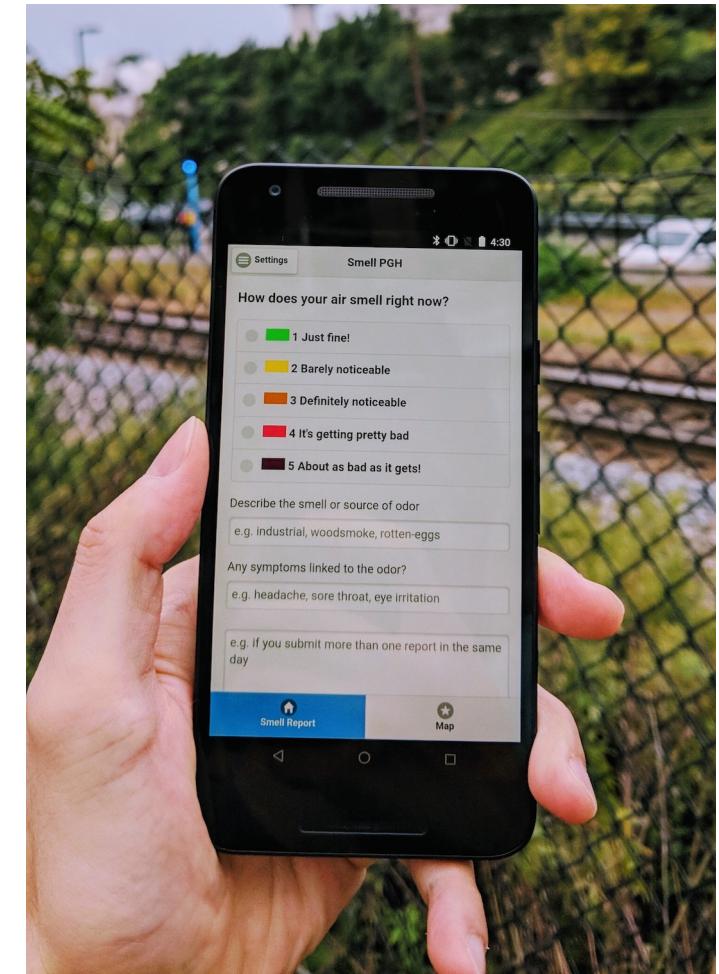
This course will assume that someone already collected the data for you. But we will briefly cover crowdsourcing and data annotation techniques.

The screenshot shows the GGD Amsterdam Data Portal. At the top, there are three small images of Amsterdam: a cityscape, a canal scene, and a street view. Below these are three sections with titles and descriptions:

- 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", "Colofon", and "Volg ons" (Twitter, Facebook, LinkedIn).

The screenshot shows the Prolific tool interface. It features a large central area with a blue background and white text. At the top, it says "Set your study live to thousands of reliable participants in minutes." Below that is a "Get started" button. In the center, there's a box for a study titled "Testing your memory of a crime scene" with details: £7.50/hour, 7 mins, 500 places. Below this are three cards for studies in Chicago, United States (Time taken: 8 mins), London, United Kingdom (Time taken: 10 mins), and Washington, United States. The bottom of the screen has a navigation bar with icons for Settings, Smell PGH, and Map.



[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 using the Python pandas, which is a very handy library for transforming and filtering structured data.

The screenshot shows a web browser displaying the "10 minutes to pandas" guide from the pandas documentation. The page has a dark theme with light-colored text and code snippets. On the left, there's a sidebar with a list of topics. The main content area features a large title, a brief introduction, and two code examples in a terminal-like interface. A navigation bar at the top includes links for Getting started, User Guide, API reference, Development, Release notes, and a search bar. To the right, there's a sidebar titled "On this page" with a list of categories.

10 minutes to pandas

- Intro to data structures
- Essential basic functionality
- IO tools (text, CSV, HDF5, ...)
- Indexing and selecting data
- MultIndex / advanced indexing
- Merge, join, concatenate and compare
- Reshaping and pivot tables
- Working with text data
- Working with missing data
- Duplicate Labels
- Categorical data
- Nullable integer data type
- Nullable Boolean data type
- Chart visualization
- Table Visualization
- Group by, split, apply, combine

10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the [Cookbook](#).

Customarily, we import as follows:

```
In [1]: import numpy as np  
In [2]: import pandas as pd
```

Object creation

See the [Intro to data structures section](#).

Creating a `Series` by passing a list of values, letting pandas create a default integer index:

```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])  
In [4]: s
```

[Show Source](#)

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.

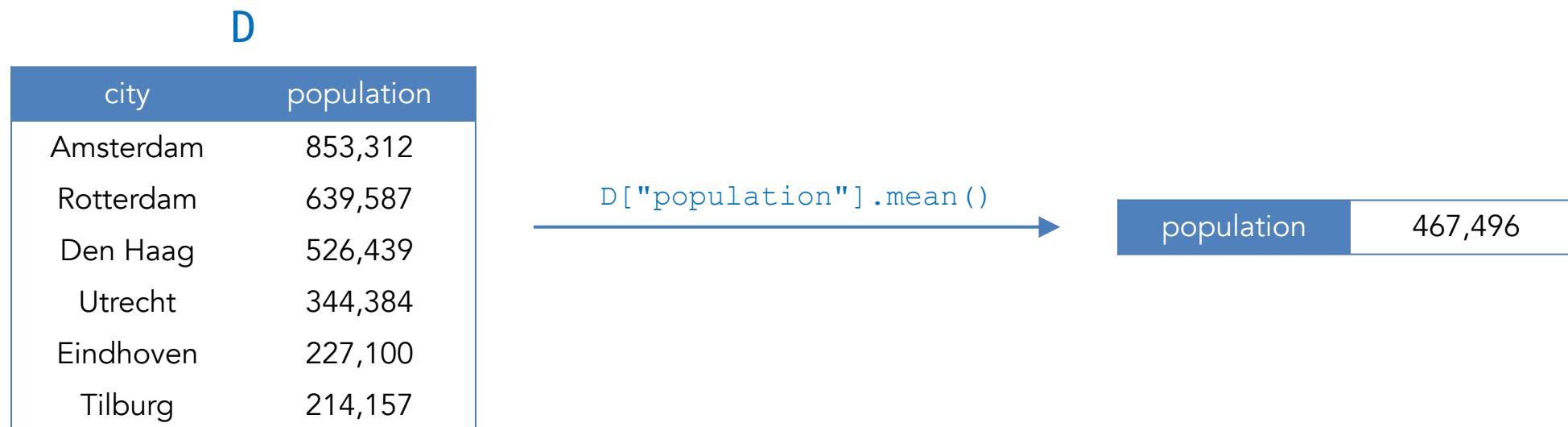
D

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	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.

D

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

`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

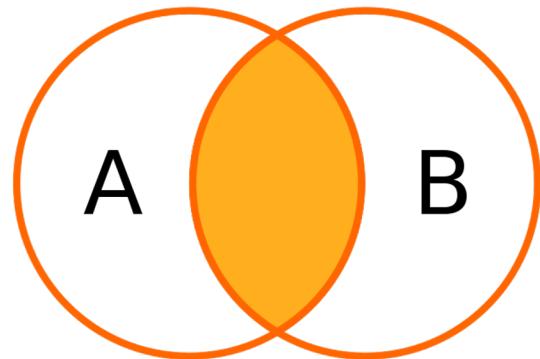
`pandas.concat([A, B])`



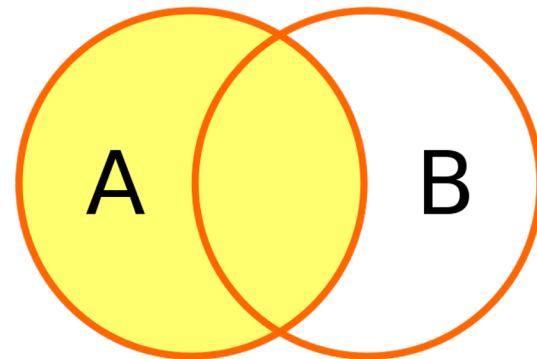
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

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

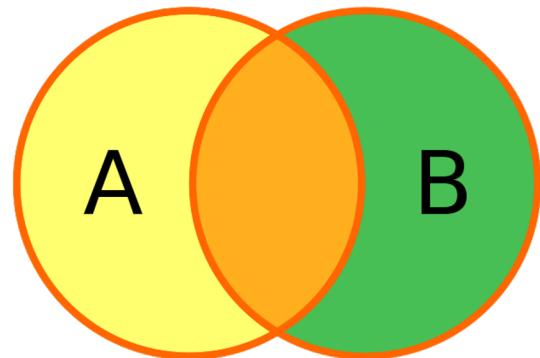
- Inner join



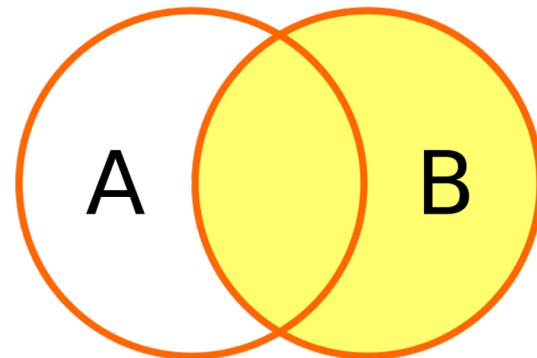
- Left (outer) join



- 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

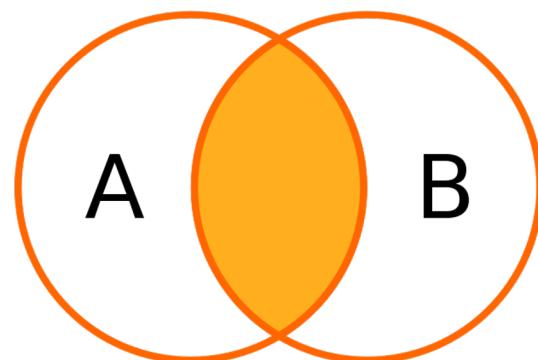
B

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

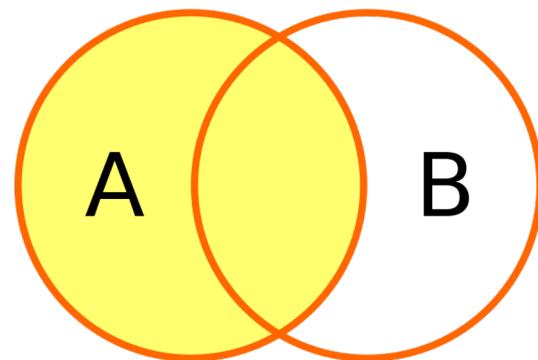
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

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 (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

A

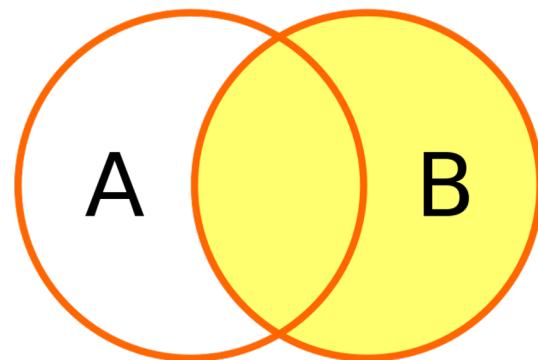
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

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 (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
Zwolle	NaN	40.9

A

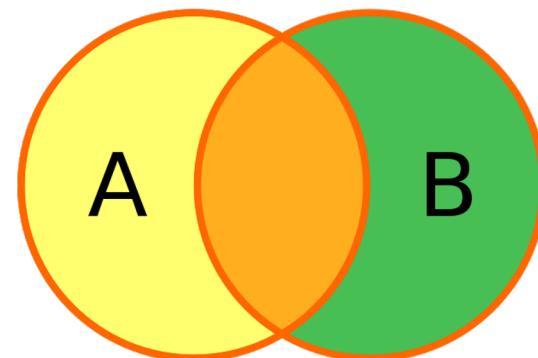
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

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

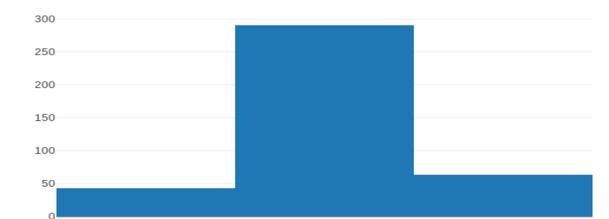
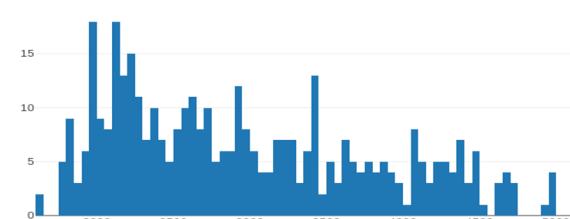
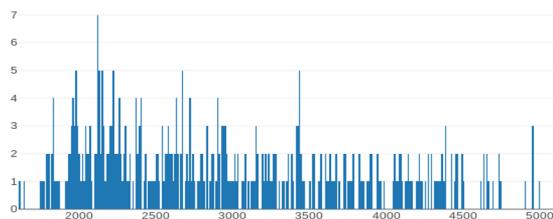
Quantization transforms a continuous set of values (e.g., integers) into a discrete set (e.g., categories). For example, age is quantized to age range.

D

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

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

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

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

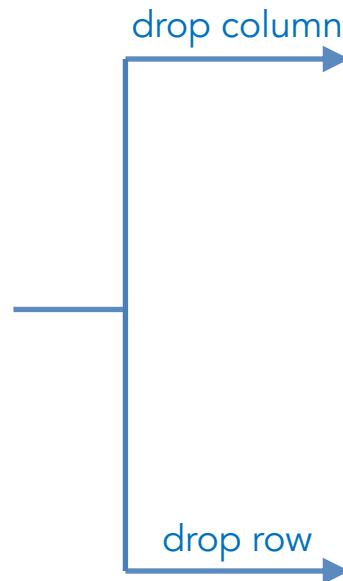
`pandas.drop([5, 6])`

	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

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	52.6
2016-10-31 08:30:00	48.3
2016-10-31 08:53:20	44.2
2016-10-31 09:30:00	31.1

`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

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

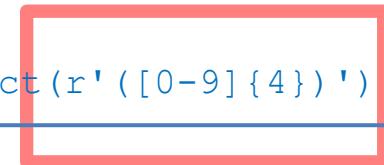
To extract data from text or match text patterns, you can use **regular expression**, which is a language to specify search patterns.

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

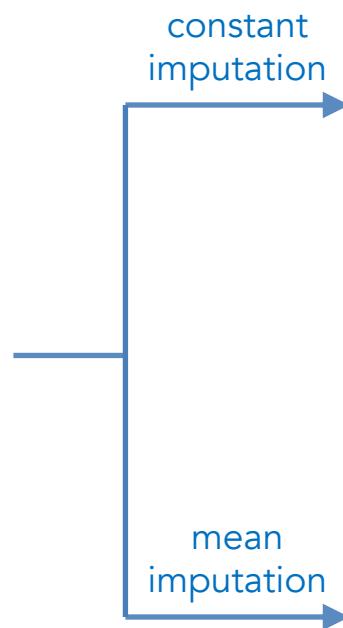
You really need to practice coding a lot to
know and internalize how these things work!

- [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](#)

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	

$$\underline{y = F(X)}$$

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.

MCAR

Missing Completely At Random:

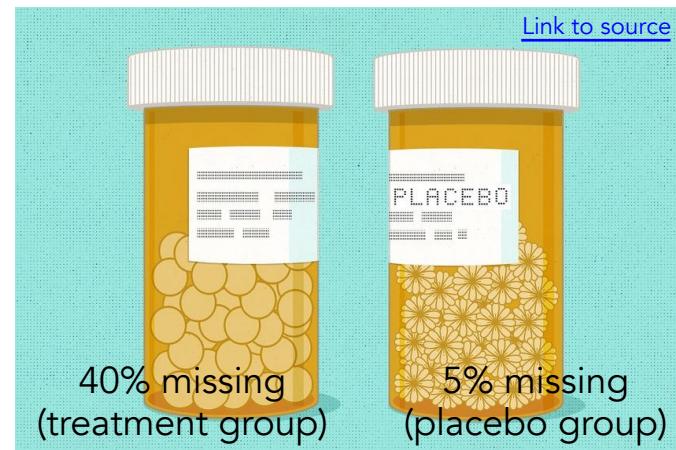
- Missing data is a completely random subset (no relations) of the entire dataset.



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)

Do you have any history of mental illness in your family? If yes, who in your family?

No

Other: _____

A survey question asking about family history of mental illness. It includes two radio button options for "No" and "Other" with a line for writing a response.

Explore Data

The course will use examples (i.e., the modules) to teach you how to explore different types of data, either using existing tools or Python.

The Netherlands in numbers 2022

How much do we cycle per week on average?

In 2020, the 12 to 17-year-olds cycled most frequently, either on bicycles or e-bikes. They also travelled the most kilometres: an average of 33 kilometres per person per week. The over-75s preferred the bicycle least. They cycled an average of 14 kilometres per week, as many as 25 to 34-year-olds.

Hoeveel fietsen we gemiddeld per week?

How much do we cycle per week on average?

Age Group	Average Distance Cycled per Week (km)
6 tot 12 jaar / 6 to 11 yrs	15,2 km
12 tot 18 jaar / 12 to 17 yrs	32,9 km
18 tot 25 jaar / 18 to 24 yrs	16,8 km
25 tot 35 jaar / 25 to 34 yrs	14,0 km
35 tot 50 jaar / 35 to 49 yrs	15,5 km
50 tot 65 jaar / 50 to 64 yrs	19,1 km
65 tot 75 jaar / 65 to 74 yrs	23,4 km
75 jaar of ouder / 75 yrs and over	13,7 km

The Netherlands in numbers 2022

Average distance from a museum is 4 km

In 2020, Dutch people lived at an average distance^① of 4.0 kilometres from a museum.^② They could choose from an average of 3.5 Dutch museums within a travel distance of five kilometres. Noord-Holland had the most museums within this distance out of all the provinces, with an average of 8.9. Within this province, residents of Amsterdam enjoyed the most choice with 23.7 museums within five kilometres, followed by Haarlem residents with 6.1 museums. Zuid-Holland also offered relatively high choice with an average of 5.1 museums within five kilometres. In that province, the highest number of museums within this distance was seen in The Hague (13.4 museums) and in Leiden (9.7 museums).

Number of Dutch museums within a 5-km radius by road, 2020

Fewer than 1
1 to 2
2 to 3
3 to 4
4 or more

Show datable

The Netherlands in numbers 2022

How much fish is landed?

In 2021, the amount of fish brought into Dutch fishery ports from sea was 443.5 million kilograms. That is 17 percent more than in the previous year. Most of the fish, 91 percent, landed frozen via trawlers, with the rest landed fresh by cutters. Blue whiting was the most landed fish at 135.9 million kilograms, plaice the most landed fresh fish at 17.6 million kilograms.

Hoeveel vis wordt aan wal gebracht?

How much fish is landed?

Fish Species	Amount (million kg)
Blauwe whiting	135,9
Haring	131,8
Makreel	72,4
Horsmakreel	42,9
Sardines	21,2
Schol	17,6

min kg

Trawlers landed 404.1 million kilograms of fish. These fishing vessels with funnel-shaped nets concentrate on schools of fish swimming in deeper waters. In 2021, they mainly landed blue

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. 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 GitHub and Stack Overflow. On the right, there are "Contents" and "Features" sections with links to 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.

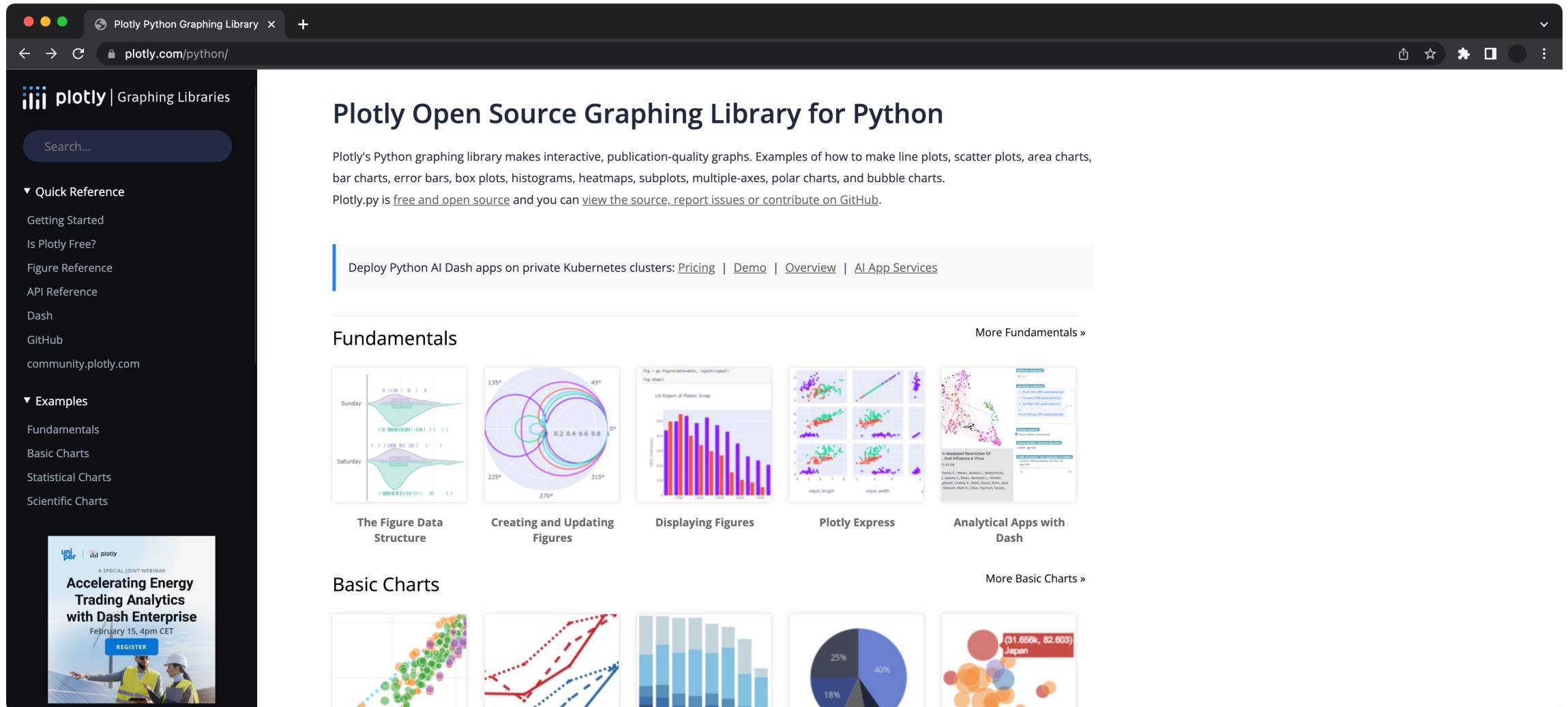
To see the code or report a bug, please visit the [GitHub repository](#). General support questions are most at home on [stackoverflow](#), which has a dedicated channel for seaborn.

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". The main content area features several examples of interactive plots, including a sunburst chart, a scatter plot with polar axes, a histogram, and a complex dashboard. A sidebar on the left contains links for "Getting Started", "Is Plotly Free?", "Figure Reference", "API Reference", "Dash", "GitHub", and "community.plotly.com". At the bottom, there's a promotional banner for a joint webinar with UniPer.

Plotly Open Source Graphing Library for Python

Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts.

Plotly.py is [free and open source](#) and you can [view the source](#), [report issues](#) or [contribute on GitHub](#).

Deploy Python AI Dash apps on private Kubernetes clusters: [Pricing](#) | [Demo](#) | [Overview](#) | [AI App Services](#)

Fundamentals

[The Figure Data Structure](#) [Creating and Updating Figures](#) [Displaying Figures](#) [Plotly Express](#) [Analytical Apps with Dash](#) [More Fundamentals »](#)

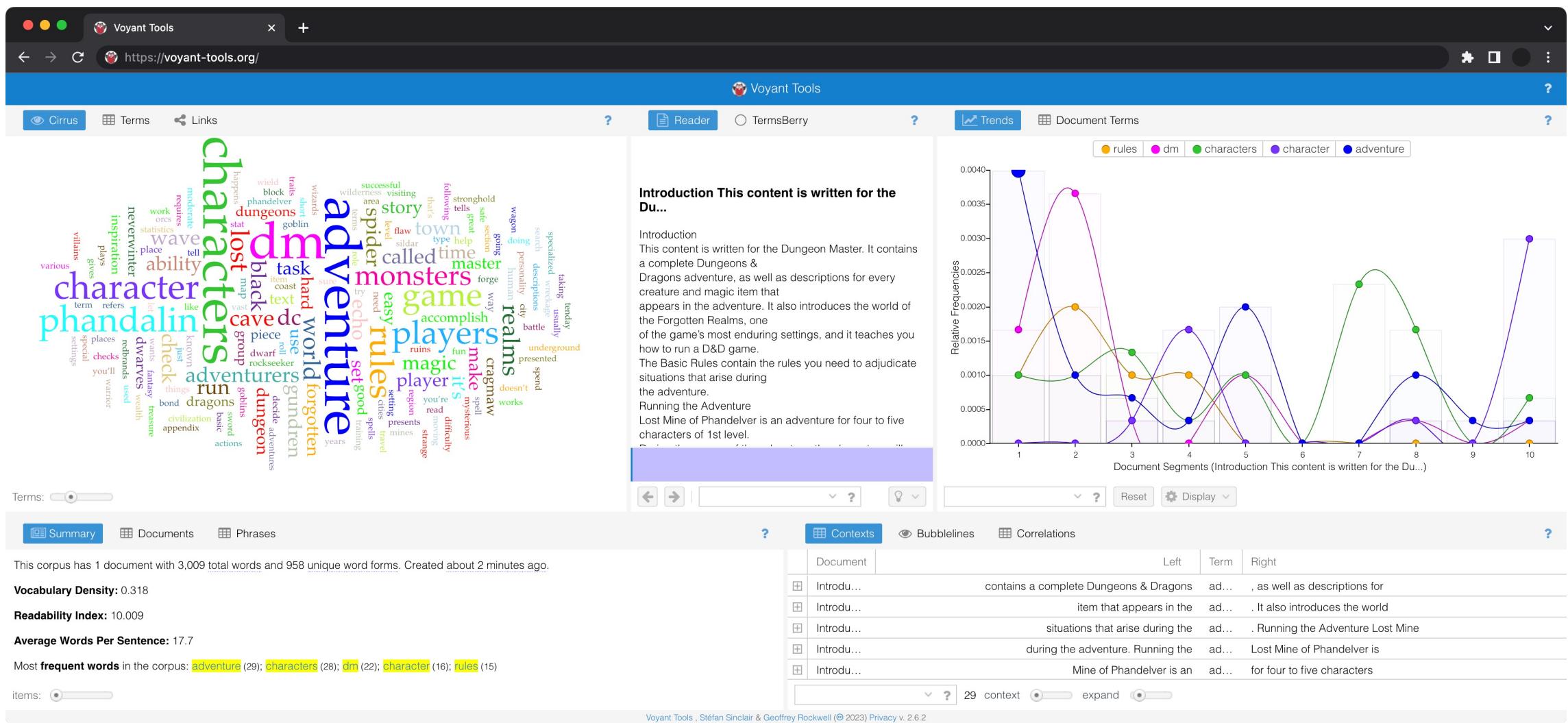
Basic Charts

[More Basic Charts »](#)

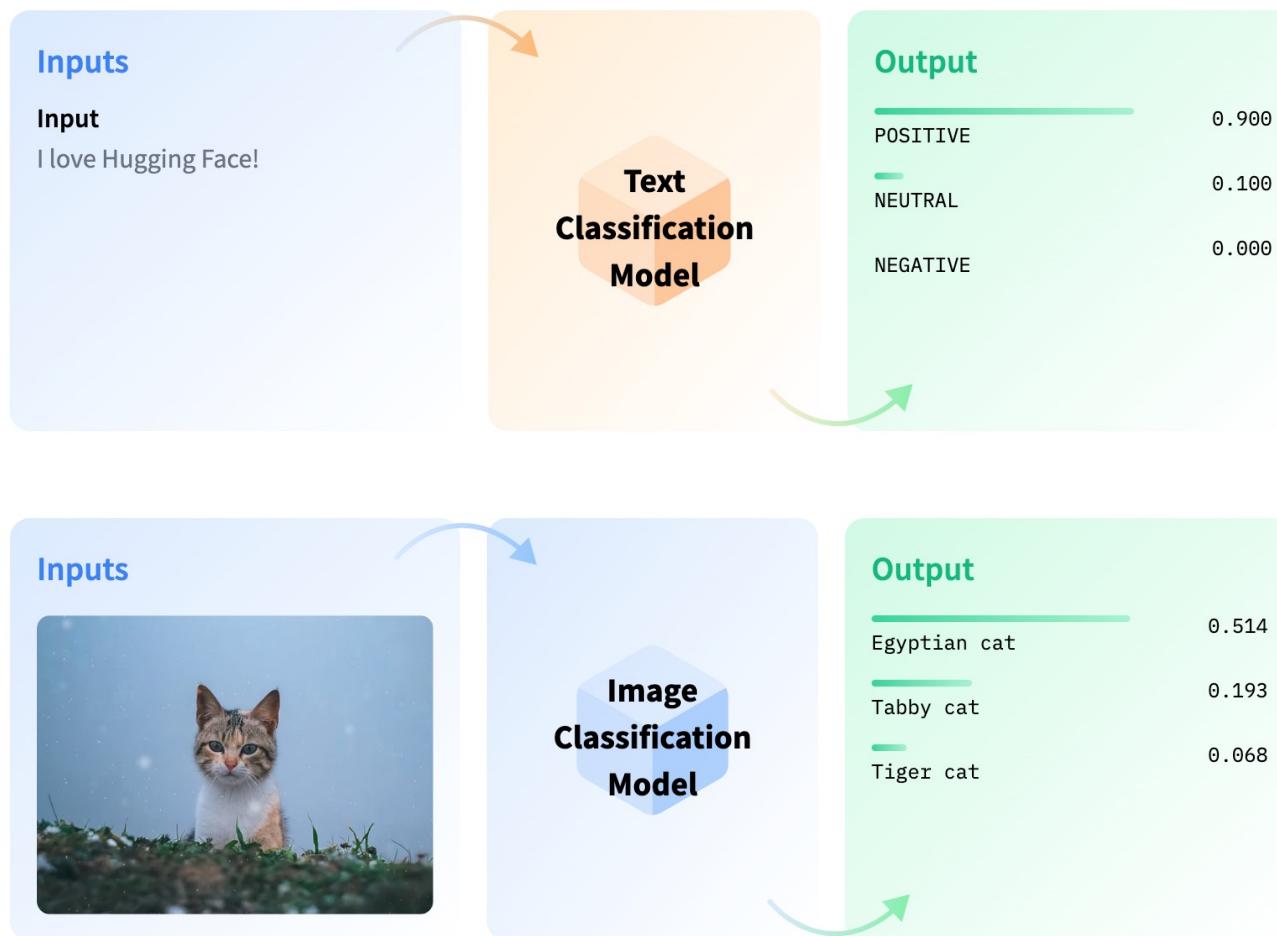
Plotly Open Source Graphing Library for Python -- <https://plotly.com/python/>

Explore Data

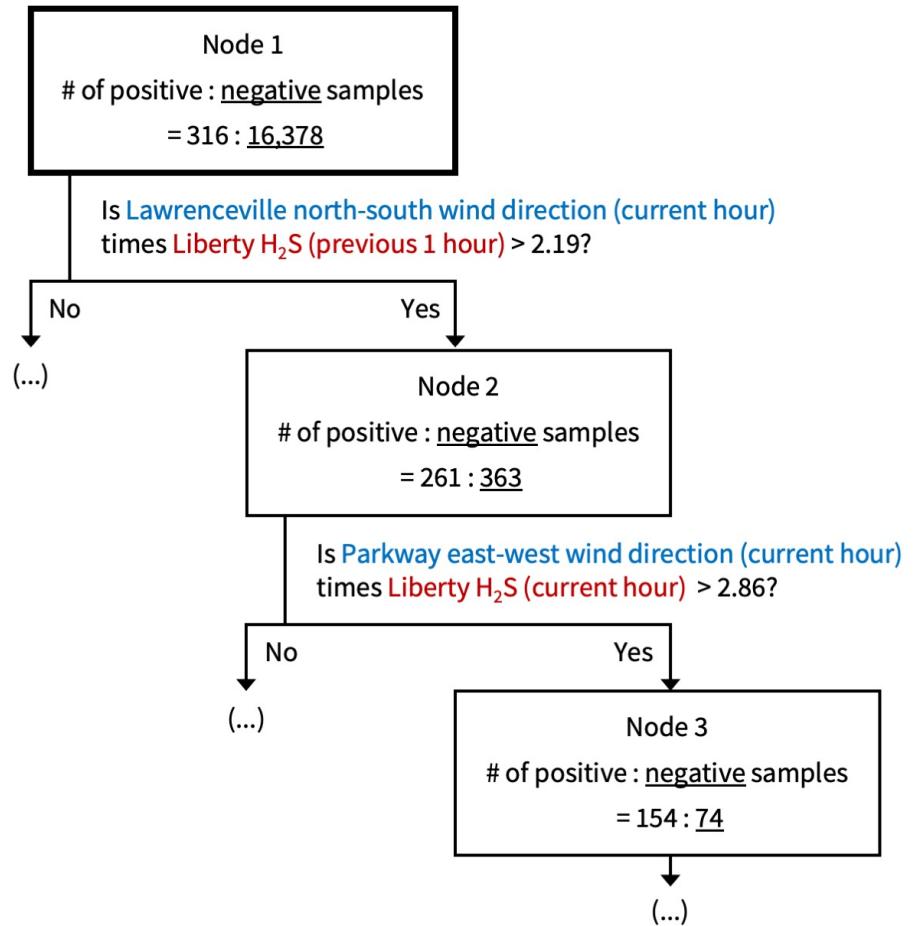
You can use the Voyant Tools to explore text data.



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

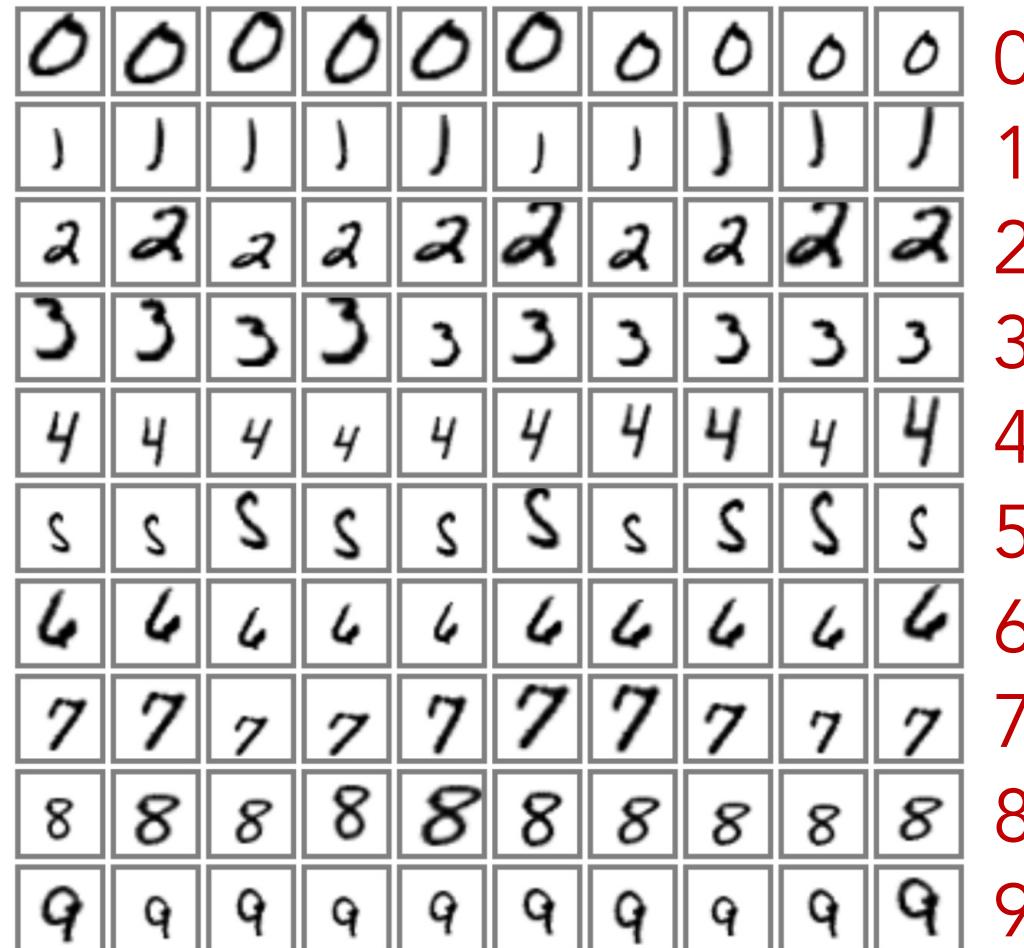


Source: <https://huggingface.co/tasks>



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 ▶

Matthew Rozsa 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.	July 27, 2018	
Leah Pickett 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.	December 26, 2017	
Jake Wilson 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...	December 16, 2017	
Peter Rainer Christian Science Monitor		Fanatics will love it; for the rest of us, it's a tolerably good time.	December 15, 2017	
Matthew Lickona San Diego Reader		The devoted will no doubt be delighted; for the rest, a resigned acceptance may be the safest path to enjoyment.	December 15, 2017	

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

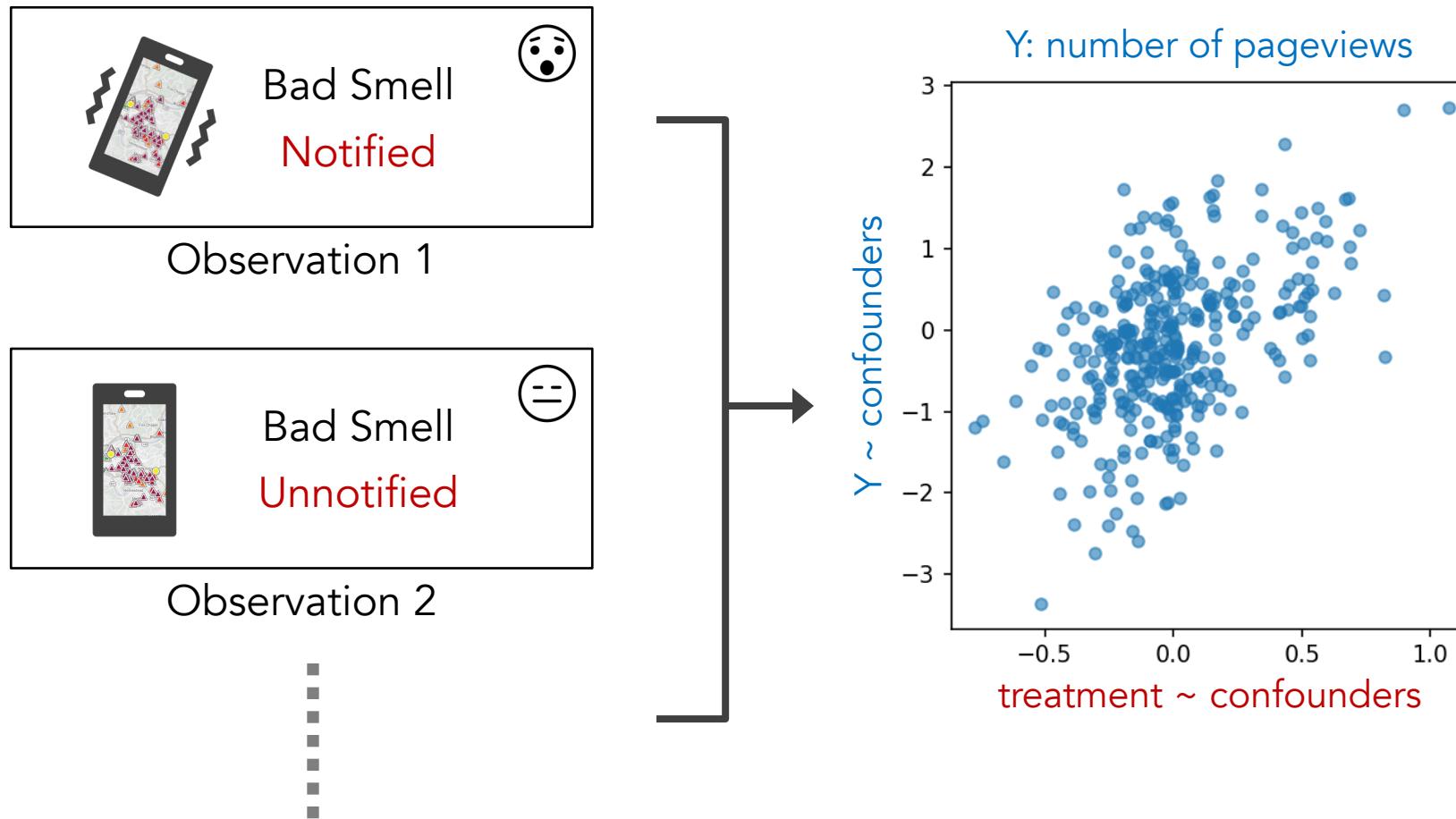
Purpose

Method

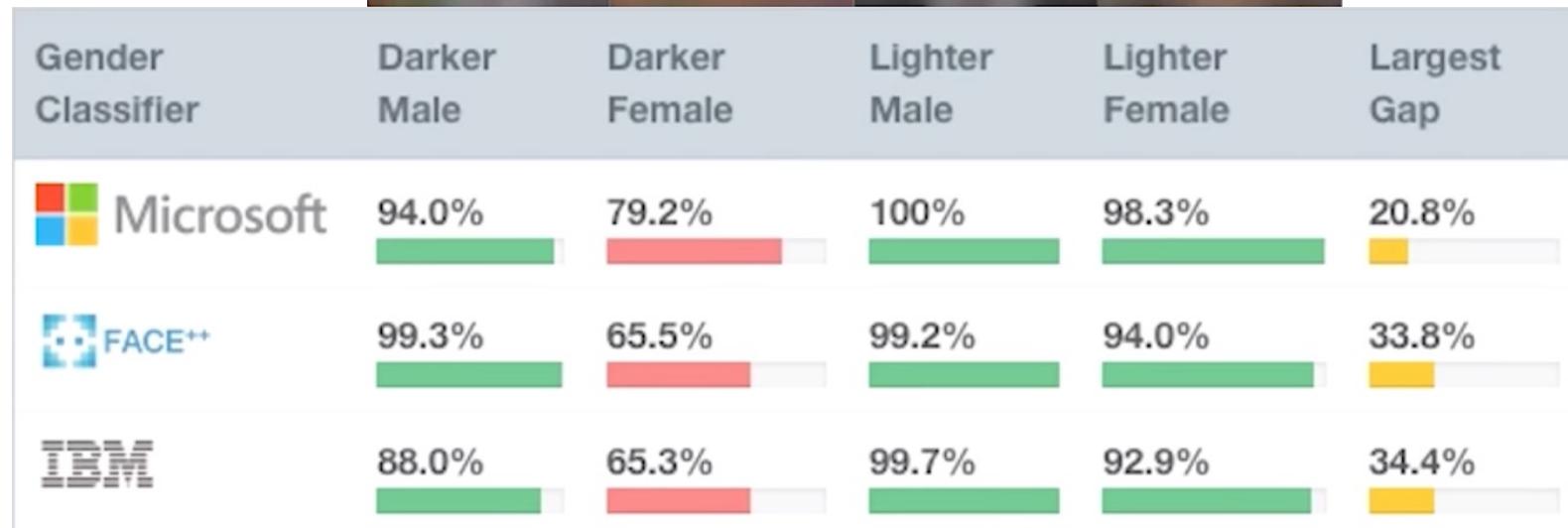
Finding

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?