VU VRIJE UNIVERSITEIT AMSTERDAM

Lecture 4: SQL and Data Wrangling

Dr. Ir. M. A. Corstanje

Recap last week

- Data engineering principles
  - Data sources, processing infrastructure, …
  - Data engineers provide businesses with the relevant data by setting up data pipelpines.

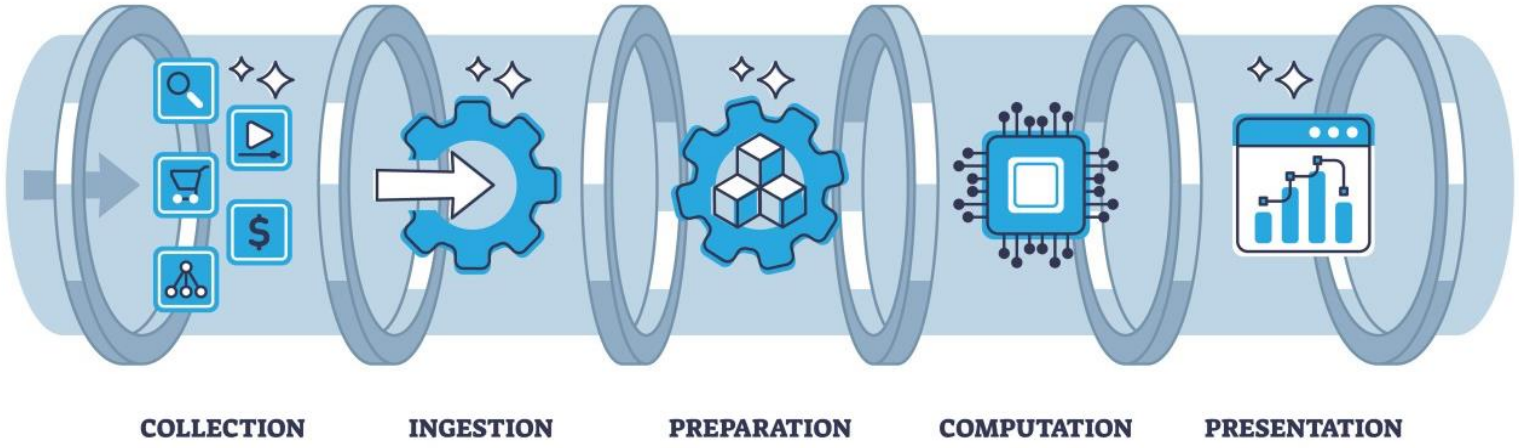
- Version control with Git
  - System that tracks changes and metadata.
  - Hosted on GitHub.
  - Very useful for software teams.

The project after today

- Working with a single `.csv` file
- After this lecture, there is an entire database availlable on canvas
- You will have to retrieve the correct data and use it in combination with the data you have

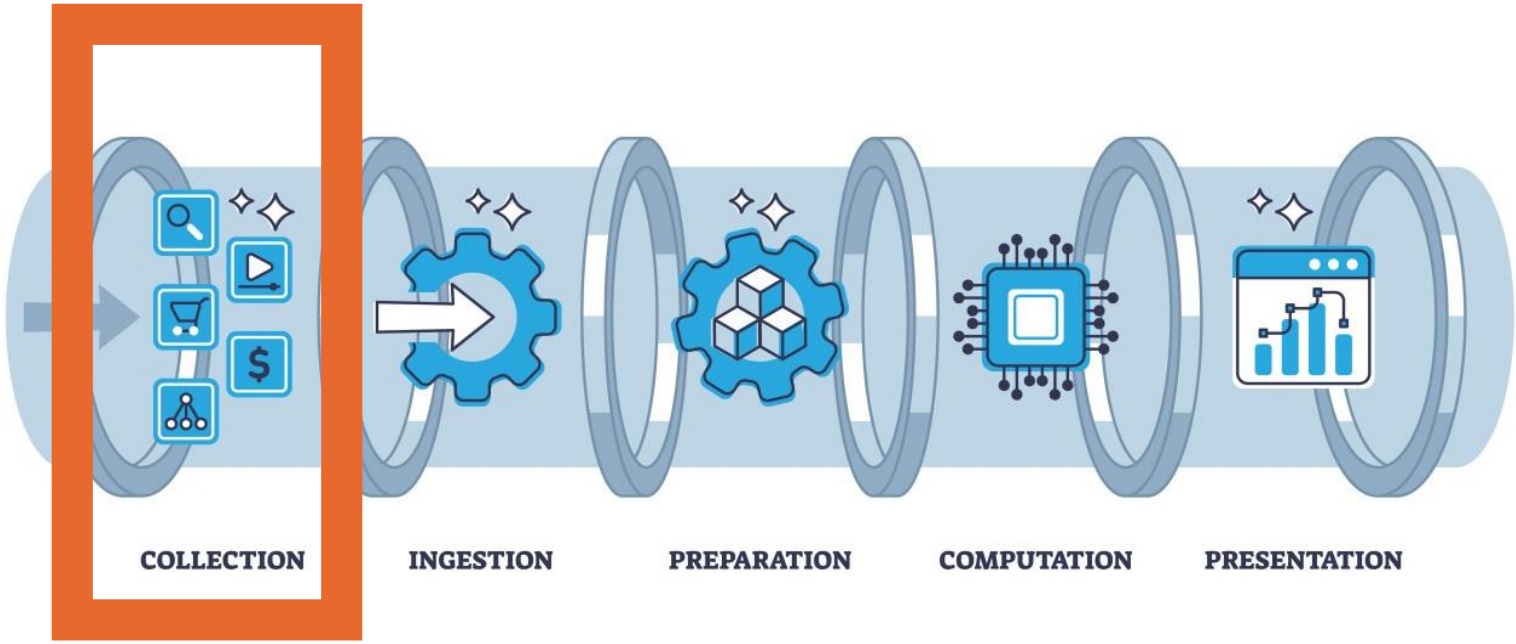
Recap data pipeline

A collection of processes for ingesting, cleaning, transforming, and storing data.



Recap data pipeline

A collection of processes for ingesting, cleaning, transforming, and storing data.



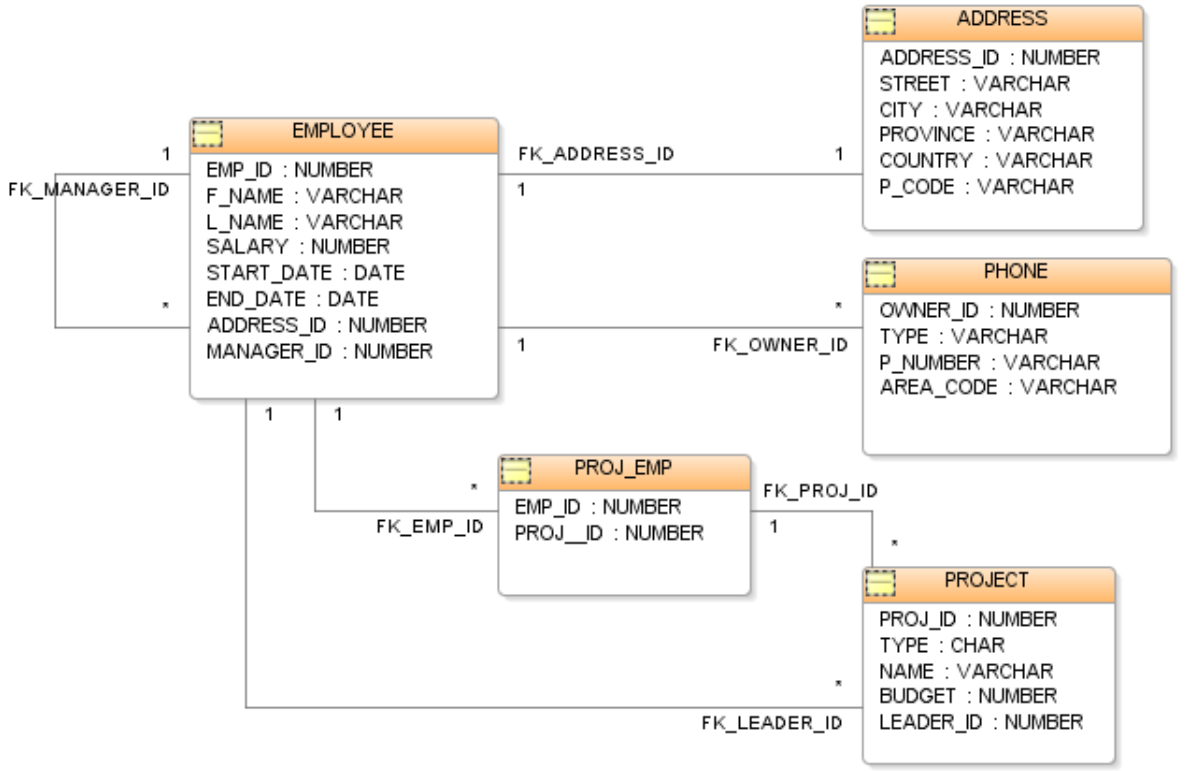
Schedule for today

- Relational databases
- Database manipulation using SQL
- Data Wrangling
  - Data acquisition from other sources

VU

Relational databases

Relational databases



- Collection of (named) tables
- Relational structure between the tables

Tables in databases

- Indexed rows
  - records
- Named columns
  - Attributes, variables, fields

	name	height	mass	hair_color	skin_color	eye_color	birth_year	sex	gender
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	masculin
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	none	masculin
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	none	masculin
4	Darth Vader	202	136.0	none	white	yellow	41.9	male	masculin
5	Leia Organa	150	49.0	brown	light	brown	19.0	female	feminine
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	masculin
7	Beru Whitesun Lars	165	75.0	brown	light	blue	47.0	female	feminine
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	none	masculin
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	masculin
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	masculin
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	masculin
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male	masculin
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male	masculin
14	Han Solo	180	80.0	brown	fair	brown	29.0	male	masculin
15	Greedo	173	74.0	*NA*	green	black	44.0	male	masculin
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphroditic	masculin
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	masculin
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	*NA*	*NA*
19	Yoda	66	17.0	white	green	brown	896.0	male	masculin
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male	masculin
21	Boba Fett	183	78.2	black	fair	brown	31.5	male	masculin

VU

Tables in databases

- Indexed rows/records
- Named columns
- `pd.DataFrame`

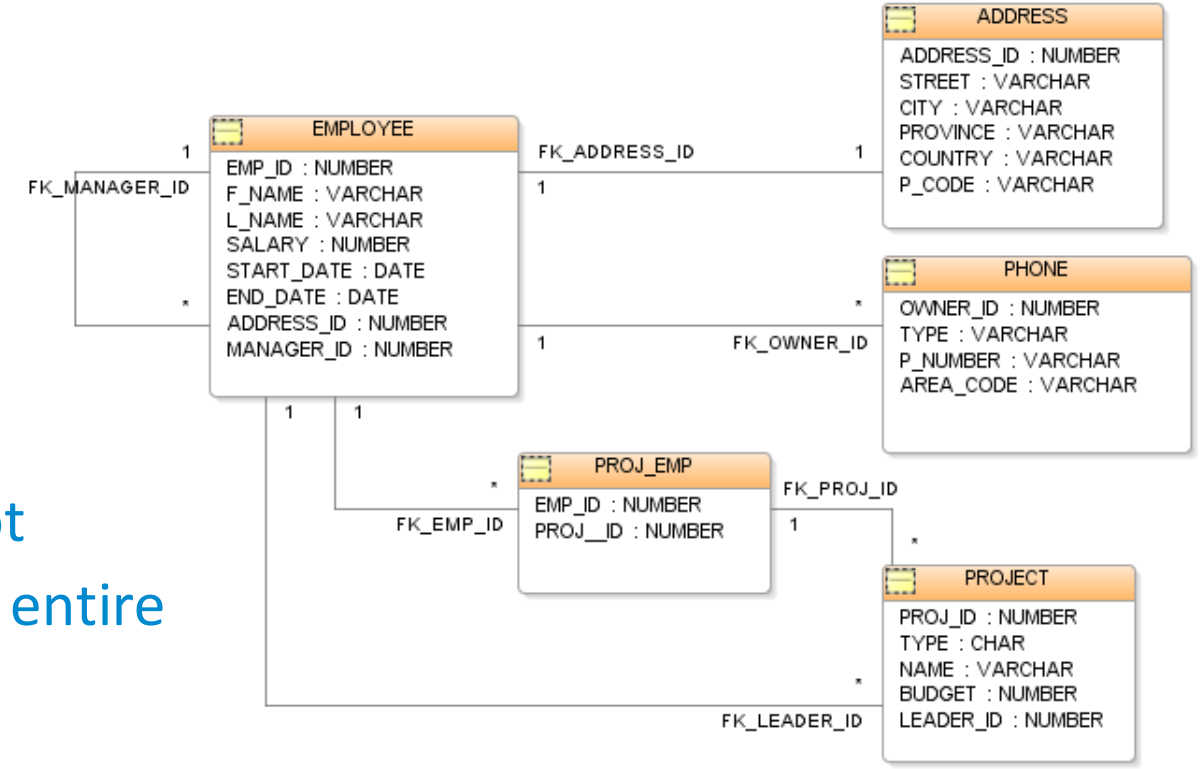
	name	height	mass	hair_color	skin_color	eye_color	birth_year	sex	gender
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	masculin
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	none	masculin
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	none	masculin
4	Darth Vader	202	136.0	none	white	yellow	41.9	male	masculin
5	Leia Organa	150	49.0	brown	light	brown	19.0	female	feminine
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	masculin
7	Beru Whitesun Lars	165	75.0	brown	light	blue	47.0	female	feminine
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	none	masculin
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	masculin
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	masculin
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	masculin
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male	masculin
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male	masculin
14	Han Solo	180	80.0	brown	fair	brown	29.0	male	masculin
15	Greedo	173	74.0	*NA*	green	black	44.0	male	masculin
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphroditic	masculin
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	masculin
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	*NA*	*NA*
19	Yoda	66	17.0	white	green	brown	896.0	male	masculin
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male	masculin
21	Boba Fett	183	78.2	black	fair	brown	31.5	male	masculin

Relational databases

- Manipulating databases means:
  - Updating records
  - Deleting/inserting records
  - Extracting (parts of the) data
  - Adding new tables
  - Etc.
  - Etc.
  - Etc.
- This is often done using SQL

Managing and retrieving data using SQL

Retrieving data



- Often, we are not interested in the entire database
- We only wish to retrieve the parts of tables we need
- Execute ``queries" against the database
  - Queries are structured requests to the database

SQL

- Structured Query Language
  - Often pronounced as ``sequel''
- Used to acces and manipulate (relational) databases
- Used when all data is stored efficiently in a centralized location
- Using SQL queries, we can do things like
  - Retrieve data from a database
  - Insert/update/delete records in data
  - In essence, a user can retrieve the data that they need

Different versions of the language

- MS SQL Server
- IBD DB2
- Oracle
- MySQL
- Microsoft Access
- SQLite

- All support the basic commands like
  - `SELECT, UPDATE, DELETE, INSERT, WHERE`

Different versions of the language

- MS SQL Server
- IBD DB2
- Oracle
- MySQL
- Microsoft Access
- SQLite

- All support the basic commands like
  - `SELECT, UPDATE, DELETE, INSERT, WHERE`

Basic commands in SQL

- `SELECT`
- `UPDATE`
- `DELETE`
- `INSERT`
- `WHERE`

SELECT

- Select prespecified columns from a table in the database.

```
SELECT column1, column2,… FROM table1;
```

SELECT

- Select prespecified columns from a table in the database.

```
SELECT column1, column2,… FROM table1;
```

- To select the entire table:

```
SELECT * FROM table1;
```

SELECT

- Select prespecified columns from a table in the database.

```
SELECT column1, column2,… FROM table1;
```

- To select the entire table:

```
SELECT * FROM table1;
```

- Variation:

```
SELECT DISTINCT column1, column2,…  FROM table1;
```


VU

WHERE

```
SELECT column1, column2,… FROM table1 WHERE condition;
```

- Select all records of prespecified columns that satisfy the `condition`.

VU

WHERE

```
SELECT column1, column2,… FROM table1 WHERE condition;
```

- Select all records of prespecified columns that satisfy the `condition`.
- Example:

```
SELECT manager_id FROM employee WHERE salary > 60000;
```

WHERE

```
SELECT column1, column2,… FROM table1 WHERE condition;
```

- Select all records of prespecified columns that satisfy the `condition`.
- Example:

```
SELECT manager_id FROM employee WHERE salary > 60000;
```

Retrieve the ids of the managers of all employees with a salay above 60,000.

UPDATE

```
UPDATE table1 SET column1 = value1, … WHERE condition;
```

- Set entries of columns that meet the `condition` to new values

DELETE

```
DELETE FROM table1 WHERE condition;
```

- Deletes records from the table that satisfy the condition

VU

INSERT

```
INSERT INTO table1 SELECT * FROM table2 WHERE condition;
```

- Insert the rows from `table2` that match the `condition` into `table1`

Create a new table

```
CREATE TABLE table_name
```

- Create a new table

Using SQL within Python

Using SQL in combination with Python

- We focus on SQLite databases.
- Once installed:

```
Import sqlite3
```

Using SQL in combination with Python

1. Connect to the database:

```
connection = sqlite3.connect("database_name.db")
```

Using SQL in combination with Python

1. Connect to the database:

```
connection = sqlite3.connect("database_name.db")
```

- Now we are connected to the database.
- We do not yet have any data, but we can run queries to retrieve it from the database

Using SQL in combination with Python

1. Connect to the database:

```
connection = sqlite3.connect("database_name.db")
```

- Now we are connected to the database.
- We do not yet have any data, but we can run queries to retrieve it from the database

- NB: This works if the database is stored locally. Make sure it is in your working directory or include a path.
- Include in the GitHub repository!

VU

Using SQL in combination with Python

2. Create a query

```
query = f"SELECT * FROM employees"
```

hint: Use f-strings to embed variable names in the string.

Using SQL in combination with Python

3. Create a cursor

In order to execute SQL statements and fetch results from SQL queries.

```
cursor = con.cursor()
cursor.execute(query)
```

With variables:

```
var = 60000
query = f"SELECT * FROM employes WHERE salary > ?"
cursor.execute(query, var)
```

Using SQL in combination with Python

3. Create a cursor

In order to execute SQL statements and fetch results from SQL queries.

```
cursor = con.cursor()
cursor.execute(query)
```

Or (if just one query)

```
cursor = connection.execute(query)
```

Using SQL in combination with Python

4. Fetch results from the query and save to a DataFrame

```
rows = cursor.fetchall()
df = pd.DataFrame(rows,
        colums = [x[0] for x in cursor.description])
```

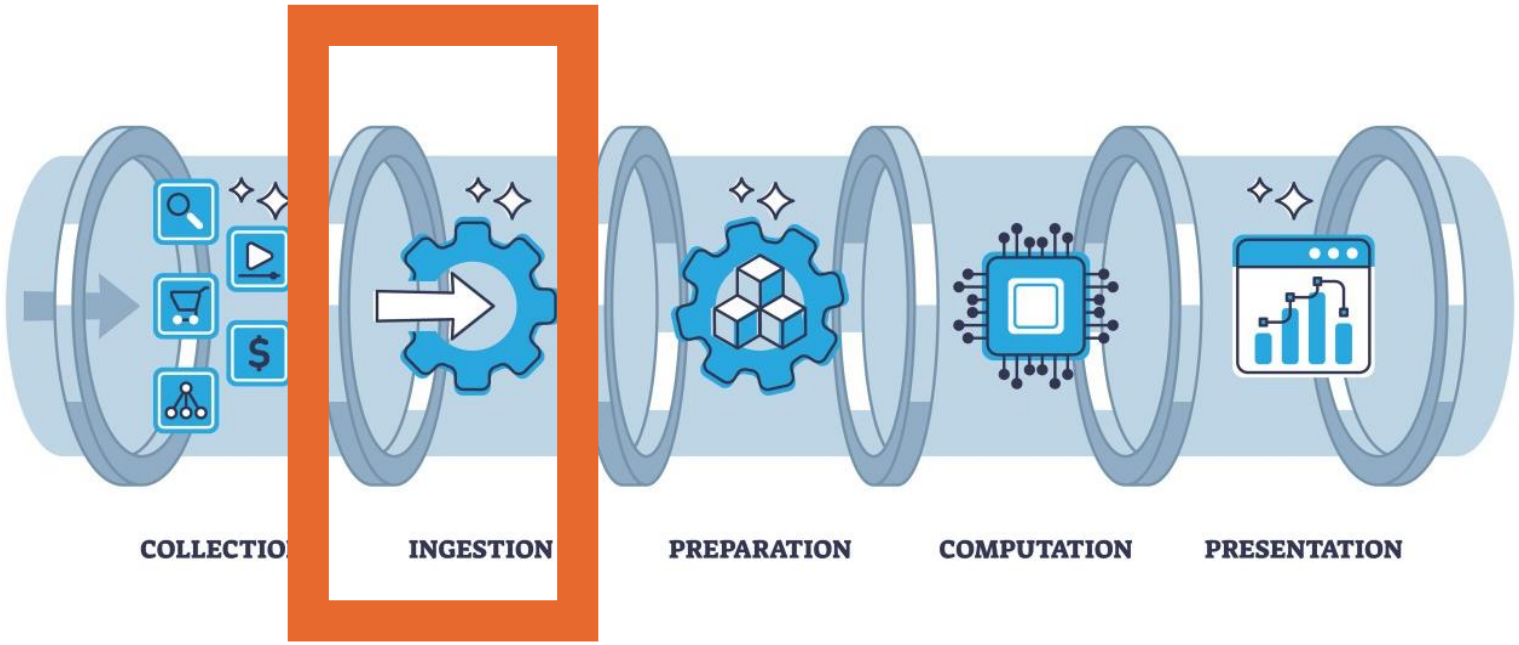

VU

Data Wrangling

Data pipeline

Data pipeline:
A collection of processes for ingesting, cleaning, transforming, and storing data.

A collection of processes for ingesting, cleaning, transforming, and storing data.



What is data wrangling

- Data acquisition
- Data cleaning
- Data merging
- Data visualization
- Data aggregation

What is data wrangling

- Data acquisition
- Data cleaning
- Data merging
- Data visualization
- Data aggregation

VU

Data acquisition

Data acquisition

- Reading `.txt` files (`.csv`, `.tsv` …)
  - `pandas.read_...()`
- Reading `.JSON` files
  - Use `JSON` or `pandas.read_json()`
- Web scraping, XML and HTML files
  - Use library `lxml` and `pandas.read_html()`
- Interacting with databases
  - SQL
- Interacting with web APIs

VU

Interacting with web APIs: example

```
import requrests
import json
url = 'https://api.github.com/repos/
        pandas-dev/pandas/issues'
req = request(url)
data = req.json()
issues = pd.DataFrame(data,
              columns = [...,…,…,…])
```


```
[
  {
    "url": "https://api.github.com/repos/pandas-dev/pandas/issues/60948",
    "repository_url": "https://api.github.com/repos/pandas-dev/pandas",
    "labels_url": "https://api.github.com/repos/pandas-dev/pandas/issues/60948/labels{/name}",
    "comments_url": "https://api.github.com/repos/pandas-dev/pandas/issues/60948/comments",
    "events_url": "https://api.github.com/repos/pandas-dev/pandas/issues/60948/events",
    "html_url": "https://github.com/pandas-dev/pandas/pull/60948",
    "id": 2857294076,
    "node_id": "PR_kwDOAA0YD86LbiPQ",
    "number": 60948,
    "title": "[backport 2.3.x] API: ignore empty range/object dtype in Index setop operations (string dtype compat) (#60
    "user": {
      "login": "jorisvandenbossche",
      "id": 1020496,
      "node_id": "MDQ6VXNlcjEwMjA0OTY=",
      "avatar_url": "https://avatars.githubusercontent.com/u/1020496?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/jorisvandenbossche",
      "html_url": "https://github.com/jorisvandenbossche",
      "followers_url": "https://api.github.com/users/jorisvandenbossche/followers",
      "following_url": "https://api.github.com/users/jorisvandenbossche/following{/other_user}",
      "gists_url": "https://api.github.com/users/jorisvandenbossche/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/jorisvandenbossche/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/jorisvandenbossche/subscriptions",
      "organizations_url": "https://api.github.com/users/jorisvandenbossche/orgs",
      "repos_url": "https://api.github.com/users/jorisvandenbossche/repos",
      "events_url": "https://api.github.com/users/jorisvandenbossche/events{/privacy}",
      "received_events_url": "https://api.github.com/users/jorisvandenbossche/received_events",
      "type": "User",
      "user_view_type": "public",
      "site_admin": false
    },
    "labels": [

    ],
    "state": "open",
    "locked": false,
    "assignee": null,
    "assignees": [

    ],
    "milestone": {
      "url": "https://api.github.com/repos/pandas-dev/pandas/milestones/119",
      "html_url": "https://github.com/pandas-dev/pandas/milestone/119",
      "labels_url": "https://api.github.com/repos/pandas-dev/pandas/milestones/119/labels",
      "id": 11466880,
      "node_id": "MI_kwDOAA0YD84ArviA",
      "number": 119,
      "title": "2.3",
      "description": "on-merge: backport to 2.3.x",
      "creator": {
        "login": "jorisvandenbossche",
        "id": 1020496,
        "node_id": "MDQ6VXNlcjEwMjA0OTY=",
        "avatar_url": "https://avatars.githubusercontent.com/u/1020496?v=4",
        "gravatar_id": "",
        "url": "https://api.github.com/users/jorisvandenbossche",
        "html_url": "https://github.com/jorisvandenbossche",
        "followers_url": "https://api.github.com/users/jorisvandenbossche/followers",
        "following_url": "https://api.github.com/users/jorisvandenbossche/following{/other_user}",
        "gists_url": "https://api.github.com/users/jorisvandenbossche/gists{/gist_id}",
        "starred_url": "https://api.github.com/users/jorisvandenbossche/starred{/owner}{/repo}",
        "subscriptions_url": "https://api.github.com/users/jorisvandenbossche/subscriptions",
        "organizations_url": "https://api.github.com/users/jorisvandenbossche/orgs",
        "repos_url": "https://api.github.com/users/jorisvandenbossche/repos",
        "events_url": "https://api.github.com/users/jorisvandenbossche/events{/privacy}",
        "received_events_url": "https://api.github.com/users/jorisvandenbossche/received_events",
        "type": "User",
        "user_view_type": "public",
        "site_admin": false
      },
      "open_issues": 49,
```


VU

Cheat sheet

Function	Description
`read_csv`	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
`read_table`	Load delimited data from a file, URL, or file-like object; use tab (`'\t'`) as default delimiter
`read_fwf`	Read data in fixed-width column format (i.e., no delimiters)
`read_clipboard`	Version of `read_table` that reads data from the clipboard; useful for converting tables from web pages
`read_excel`	Read tabular data from an Excel XLS or XLSX file
`read_hdf`	Read HDF5 files written by pandas
`read_html`	Read all tables found in the given HTML document
`read_json`	Read data from a JSON (JavaScript Object Notation) string representation
`read_msgpack`	Read pandas data encoded using the MessagePack binary format
`read_pickle`	Read an arbitrary object stored in Python pickle format
`read_sas`	Read a SAS dataset stored in one of the SAS system's custom storage formats
`read_sql`	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
`read_stata`	Read a dataset from Stata file format
`read_feather`	Read the Feather binary file format

Plan for now

Continue the projects!

- Databases are now availlable on Canvas
- Assignments part 123 are now availlable on canvas

- You are free to explore the database and look for interesting stuff

- The assignments contain suggested tasks that can be interesting
  - Depending on how tomorrow goes, I might add more or highlight a few!

- Good luck!

Data cleaning

What is data wrangling

- Data acquisition
- Data cleaning
- Data merging
- Data visualization
- Data aggregation

Examples of data cleaning

- Handling missing data
  - Filtering out missing data
  - Replacing missing values
- Data transformations
  - Removing duplicates
  - Transforming using a function
  - Replacing values
  - Replacing indexes
  - Discretization and binning
- Outlier analysis

Missing data

- NaN (Not a Number)

- Reasons for missing data?
  - Not every question in a survey filled in
  - Not everyone participated in a trial
  - Privacy
  - …

Finding missing data

```
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
display(string_data)
```

```
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object
```

```
string_data.isnull()
```

```
0    False
1    False
2     True
3    False
dtype: bool
```

```
string_data[0] = None
string_data.isnull()
```

```
0     True
1    False
2     True
3    False
dtype: bool
```

Handling missing data

- Remove the missing values using for instance `.dropna()`

- Replace using for instance
- `.ffill()`
  - `Replaces it with value above`
- `.fillna(value)`
  - `Replaces NaNs with the given value`

Transforming data – removing duplicates