

Databases

06 Data Manipulation (Python, Pandas)

Lucas Iacono

Know-Center & Graz University of Technology, Austria

Computer Science and Biomedical Engineering
Institute of Human-Centred AI

Course calendar

- **Part 2: Data handling and project presentations**
 - **(5) Mon. 01.12.2025:** Data handling and manipulation with Python (Pandas)
 - **Mon. 08.12.2025:** public holiday (means no lecture – **time for exercise 2**)
 - **Exercise 2 submission: Fri. 12.12.2025**
 - **Mon. 15.12.2024:** No lecture (time for group project full submission)
 - **Project submission: Fri. 19.12.2025**
 - **(6) Mon. 12.01.2026:** project discussions of selected groups
 - **(7) Mon. 19.01.2026:** project discussions of selected groups 2 [if needed]
 - **(8) Mon. 26.01.2026:** No lecture: grading

Last lecture

- **Query Languages**

- Structured Query Language (SQL)
 - Data Definition Language → Schema creation, schema modification
 - Data Manipulation Language → How to query your DB, answer questions about data
 - Data Control Language (just a little bit) → Assign/Change permissions

This week: Agenda

- **Data Manipulation (Python, Pandas, Matplotlib)**
 - Overview about APIs and Psycopg2
 - Code examples →
 - Jupyter notebook is **available in TeachCenter**
 - **Also backup of a movie database → create database →right-click on it → “restore” (choose .sql backup file)**
- **Discussion about Groups Presentation**
- **Important:**
 - **Use correct file names! (points deductions)**
 - **Ex1 & Ex2 → Individually! (plagiarism)**

Application Programming Interface (API)

What's an API and why should we care?

- **Application Programming Interface (API)**
 - Defined **set of functions or protocols** for system or component communication
 - Interface independent of concrete implementation → **decoupling of applications** from underlying libraries / systems
 - API stability of utmost importance

What's an API and why should we care?

- **Examples**

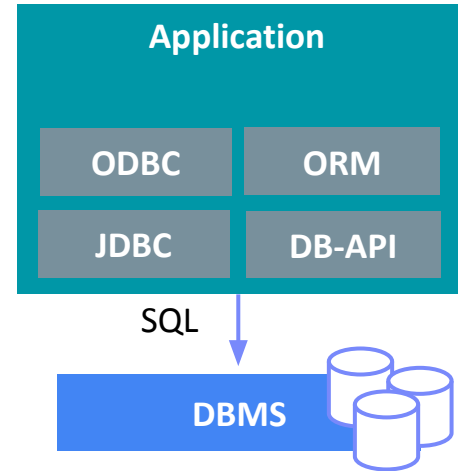
- **Linux:** kernel-user space API → system calls, POSIX (Portable Operating System Interface)
- **Cloud Services:** often dedicated REST (Representational State Transfer) APIs

What's an API and why should we care?

- **Examples**

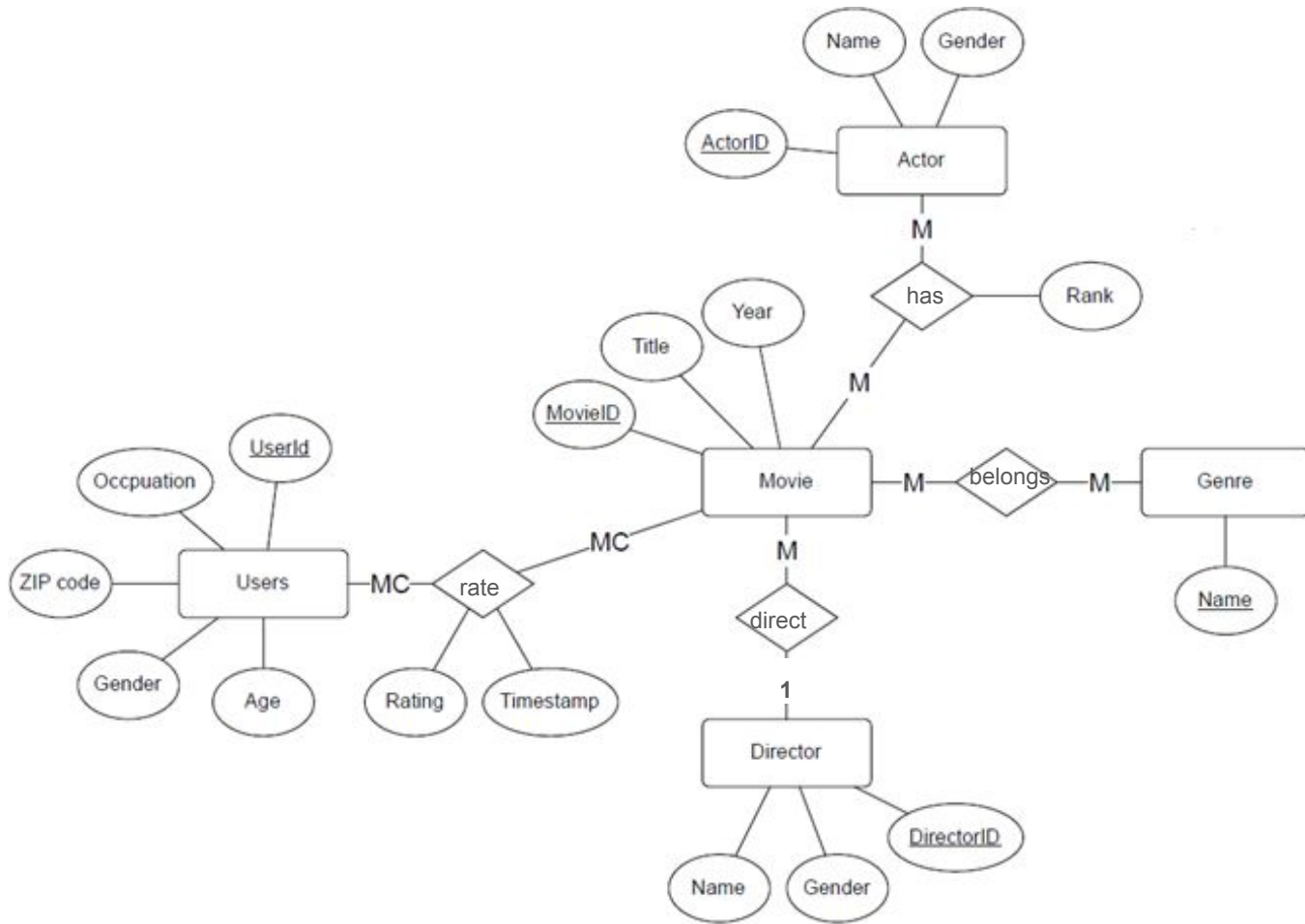
- **DB Access:**

- ODBC: C/C++
- JDBC: Java
- **DB-API 2.0 = standard that defines how libraries should work to connect to databases in Python**
- ORM: maps programming language objects to database structures → work with objects instead of SQL and change databases without altering application code.



Movie Database Setting (for Python code examples)

ER



Schema (1)

```

CREATE TABLE Users (
  UserID INT PRIMARY KEY,
  Age INT,
  Gender VARCHAR(1), -- or VARCHAR (256)
  Occupation VARCHAR(256),
  ZIP VARCHAR(5)
);

CREATE TABLE Actor (
  ActorID INT PRIMARY KEY,
  Name VARCHAR (256),
  Gender VARCHAR (256),
  Ranking INT
);

CREATE TABLE Genre (
  GenreID INT PRIMARY KEY, -- not obligatory surrogate key
  Name VARCHAR (256) --can also be chosen as a PK
);

```

Schema (2)

```

CREATE TABLE Director (
  DirectorID INT PRIMARY KEY,
  Name VARCHAR(256),
  Gender VARCHAR(256)
);

CREATE TABLE Movie (
  MovieID INT PRIMARY KEY,
  Title VARCHAR(256),
  Year INT,
  DirectorID INT REFERENCES Director (DirectorID)
);

CREATE TABLE UserMovie (
  UserID INT REFERENCES Users (UserID),
  MovieID INT REFERENCES Movie (MovieID),
  Rating Decimal(2,1), -- or Numeric(2,1)
  TS Timestamp, -- or INT (because we have Unix-timestamps)
  PRIMARY KEY (UserID,MovieID)
);

CREATE TABLE ActorMovie (
  ActorID INT REFERENCES Actor (ActorID),
  MovieID INT REFERENCES Movie (MovieID),
  Rank INT,
  PRIMARY KEY (ActorID,MovieID)
);

CREATE TABLE MovieGenre (
  GenreID INT REFERENCES Genre (GenreID),
  MovieID INT REFERENCES Movie (MovieID),
  PRIMARY KEY (GenreID,MovieID)
);

```

Psycopg2

Psycopg (Python PostgreSQL Adapter)

- **Overview Psycopg**

- Implements **Python Database API Specification v2.0** (PEP 248/249 in 2001)
- Install via **pip install psycopg2** (or **conda install conda-forge::psycopg**)
 - Pip comes with standard Python installation: <https://www.python.org/downloads/>
 - Conda via Anaconda installation: <https://www.anaconda.com/>

Psycopg (Python PostgreSQL Adapter)

- **Overview Psycopg**

- Implements **Python Database API Specification v2.0** (PEP 248/249 in 2001)
- Install via **pip install psycopg2** (or **conda install conda-forge::psycopg**)
 - Pip comes with standard Python installation: <https://www.python.org/downloads/>
 - Conda via Anaconda installation: <https://www.anaconda.com/>

- **Establish Connection**

```
import psycopg2
```

```
conn = psycopg2.connect(  
    host="localhost", port="5432",  
    database="db1234567", user=username,  
    password=password)
```

Psycopg (Python PostgreSQL Adapter)

- **Execute Statements**

- Use local cursors
- Commit to persist

```
cur = conn.cursor()
cur.execute("INSERT INTO Students VALUES(...)")
conn.commit()
```

- **Process Result Sets**

- Access properties via indices

```
cur.execute("SELECT SID, LName FROM Students")
students = cur.fetchall() #get all rows returned by SELECT.
for row in students:
    print("SID = ", row[0], end = ',')
    print("Lname = ", row[1])
```

Recap: Psycopg (Python PostgreSQL Adapter)

- **Execute Prepared Statements**

- Good for performance
- Prevents SQL injection

```
cur = conn.cursor()
sql = "INSERT INTO Students VALUES(%s, %s)"
for s in students:
    cur.execute(sql, (s.getID(),s.getName()))
conn.commit()
```

- **Execute Callable Statement**

- Stored procedures

```
cur = conn.cursor()
cur.callproc("prepStudents", (2019, 2))
rows = cur.fetchone()
```

- **Close Connection**

- Performance and Integrity!

```
cur.close()
conn.close()
```

Recap: Psycopg (Python PostgreSQL Adapter)

- **Execute Prepared Statements**

- Good for performance
- Prevents SQL injection

```
cur = conn.cursor()
sql = "INSERT INTO Students VALUES(%s, %s)"
for s in students:
    cur.execute(sql, (s.getID(),s.getName()))
conn.commit()
```

- **Execute Callable Statement**

- Stored procedures

```
cur = conn.cursor()
cur.callproc("prepStudents", (2019, 2))
rows = cur.fetchone()
```

- **Close Connection**

- Performance and Integrity!

```
cur.close()
conn.close()
```

Code Example

Data Handling and Manipulation with Pandas (Basics)

Pandas – Short Introduction

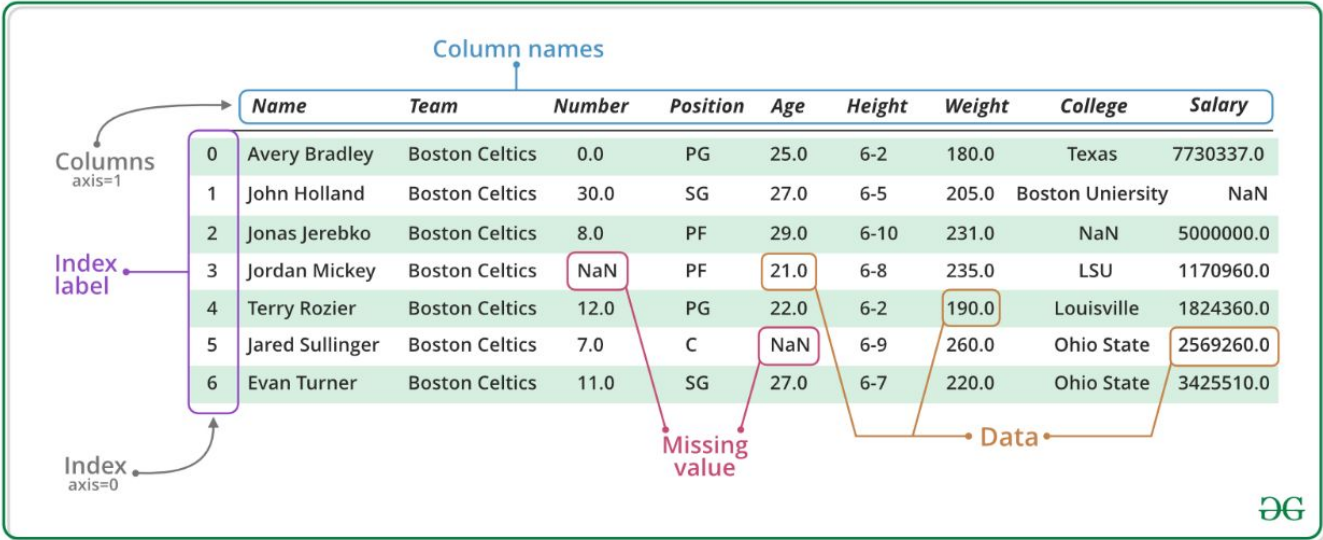
- **Pandas**

- Data analysis and manipulation library for Python
- Install via **pip install pandas** (already comes with anaconda)

- **DataFrame**

- Similar to a database table

[wikimedia.org]



The diagram shows a DataFrame table with the following structure and annotations:

- Column names:** Name, Team, Number, Position, Age, Height, Weight, College, Salary.
- Index labels:** 0, 1, 2, 3, 4, 5, 6.
- Missing value:** A pink box highlights the 'NaN' value in the 'Number' column for index 3.
- Data:** Orange boxes highlight specific data points: '21.0' (Age), '190.0' (Weight), '2569260.0' (Salary), and '3425510.0' (Salary).
- Axis labels:** 'Columns axis=1' points to the column headers, and 'Index axis=0' points to the row indices.

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Psycopg (Work with Pandas)

- **Setting**

- PostgreSQL **movie** database with “movies” table
- Python object **movies** → SELECT * FROM movies;

- **ResultSet to DataFrame**

```
df_movies = pandas.DataFrame(movies)
```

Psycopg (Work with Pandas)

- **Get/set column names**

```
colnames = [d[0] for d in cur.description]
df_movies.columns = colnames
```

- **Set index**

```
df_movies = df_movies.set_index('movieid')
```

- **Do queries etc.**

```
print(df_movies.query('year > 2007'))
```

- **Access by id, attribute or value**

```
movie = df_movies.loc[58293]
```

```
movie = df_movies.loc[:, "year"]
```

```
movie = df_movies.loc[
    df_movies['title'] == 'Ninja Assassin']
```

Psycopg (Work with Pandas)

- Mean(), min(), max()...

```
df_movies['year'].min()
df_movies['year'].max()
```

- Describe()

- Get statistics of all numeric columns

```
df_movies.describe()
```

	year
count	345.000000
mean	2008.095652
std	0.340329
min	2008.000000
25%	2008.000000
50%	2008.000000
75%	2008.000000
max	2011.000000

- Sorting

- Ascending, descending

```
df_movies.sort_values('title')
```

Psycopg (Work with Pandas)

- Mean(), min(), max()...

```
df_movies['year'].min()
df_movies['year'].max()
```

- Describe()

- Get statistics of all numeric columns

```
df_movies.describe()
```

	year
count	345.000000
mean	2008.095652
std	0.340329
min	2008.000000
25%	2008.000000
50%	2008.000000
75%	2008.000000
max	2011.000000

- Sorting

- Ascending, descending

```
df_movies.sort_values('title')
```

Code Example

Questions / Comments?

Pandas (Merge dataframes)

- **Similar to db joins**
 - Inner, left outer, right outer join, etc.
- **Setting**
 - Same **df_movies** as before
 - Merge with **df_genres** (SELECT * FROM genres;)

movieid	title	year
58293	10,000 BC	2008
58803	21	2008
56949	27 Dresses	2008
1752	A Hard Rain's Gonna Fall	2008
5471	A Perfect Getaway	2009

movieid	name
0	1 Adventure
1	1 Animation
2	1 Children
3	1 Comedy

Pandas (Merge dataframes)

- **Pandas.merge**
 - Left df, right df, on (which foreign key column?), how (which kind of join?)

```
df_movies_genres = pd.merge(df_movies, df_genres, on='movieid', how='inner')
df_movies_genres.columns =
    ['movieid', 'movietitle', 'movieyear', 'moviecritics', 'genrename']
```

	movieid	movietitle	movieyear	moviecritics	genrename
0	1	Toy story	1995	9	Adventure
1	1	Toy story	1995	9	Animation
2	1	Toy story	1995	9	Children
3	1	Toy story	1995	9	Comedy
4	1	Toy story	1995	9	Fantasy

Pandas (Merge dataframes)

- **Access data in merged dataframe**
 - Same syntax as for a “normal” dataframe

- **Get unique genres**
 - How many?

```
df_movies_genres['genrename'].unique()  
len(...)
```

- **Get genres of “Toy story”**

```
df_movies_genres.query  
( 'movietitle == "Toy story" ' )['genrename']
```

Pandas (Group-by & distributions)

- **Group-by**

- Same as in SQL
- How many genres per movie?

```
movies =  
df_movies_genres.groupby('movieid')
```

Pandas (Group-by & distributions)

- **Group-by**

- Same as in SQL
- How many genres per movie?
- **m[0]** is the **movieid**

```
movies =  
df_movies_genres.groupby('movieid')  
  
for m in movies:  
    len(m[1]['genrename'])
```

Pandas (Group-by & distributions)

- **Group-by**

- Same as in SQL
- How many genres per movie?
- `m[0]` is the `movieid`

```
movies =
df_movies_genres.groupby('movieid')
```

```
for m in movies:
    len(m[1]['genrename'])
```

- **Value-counts**

- Creates a distribution
- How often is each genre used?

```
df_movies_genres['genrename']
.value_counts()
```

Pandas (Group-by & distributions)

- **Group-by**

- Same as in SQL
- How many genres per movie?
- `m[0]` is the `movieid`

```
movies =
df_movies_genres.groupby('movieid')
```

```
for m in movies:
    len(m[1]['genrename'])
```

- **Value-counts**

- Creates a distribution
- How often is each genre used?

```
df_movies_genres['genrename']
.value_counts()
```

```
Drama      5076
Comedy     3566
Thriller   1664
Romance    1644
Action     1445
```

Pandas (Group-by & distributions)

- **Group-by**

- Same as in SQL
- How many genres per movie?
- `m[0]` is the `movieid`

```
movies =
df_movies_genres.groupby('movieid')
```

```
for m in movies:
    len(m[1]['genrename'])
```

- **Value-counts**

- Creates a distribution
- How often is each genre used?

```
df_movies_genres['genrename']
.value_counts()
```

Drama	5076
Comedy	3566
Thriller	1664
Romance	1644
Action	1445

Code Example

Questions / Comments?

Reference:

https://pandas.pydata.org/docs/user_guide/index.html

Data Visualization with Matplotlib (Basics)

Matplotlib – Short Introduction

- **Matplotlib**
 - Data visualization library for Python
 - Install via `pip install matplotlib` (already comes with anaconda)

Matplotlib – Short Introduction

- **Matplotlib**

- Data visualization library for Python
- Install via `pip install matplotlib` (already comes with anaconda)

- **Examples of plot types**

- Pairwise data (x, y): line plot, scatter plot, bar, ...
- Statistical distributions: histogram, boxplot, pie, ...
- Others: 3D plots, ...

Matplotlib – Short Introduction

- **Matplotlib**

- Data visualization library for Python
- Install via `pip install matplotlib` (already comes with anaconda)

- **Examples of plot types**

- Pairwise data (x, y): line plot, scatter plot, bar, ...
- Statistical distributions: histogram, boxplot, pie, ...
- Others: 3D plots, ...

- **Reference**

- https://matplotlib.org/stable/users/explain/quick_start.html

Matplotlib - Examples

- **Value-counts**

- Creates a distribution
- How often is each genre used?

```
genre_dist = df_movies_genres['genrename']
.value_counts()
```

Drama	5076
Comedy	3566
Thriller	1664
Romance	1644
Action	1445

Matplotlib - Examples

- **Value-counts**

- Creates a distribution
- How often is each genre used?

```
genre_dist = df_movies_genres['genrename']
.value_counts()
```

Drama	5076
Comedy	3566
Thriller	1664
Romance	1644
Action	1445

- **Simple line plot**

- `matplotlib.plot(genre_dist)`
- Set `x_label`, `y_label`, `title`, `font_sizes`, etc.
- Save figure as pdf, png, etc.

Matplotlib - Examples

- **Value-counts**

- Creates a distribution
- How often is each genre used?

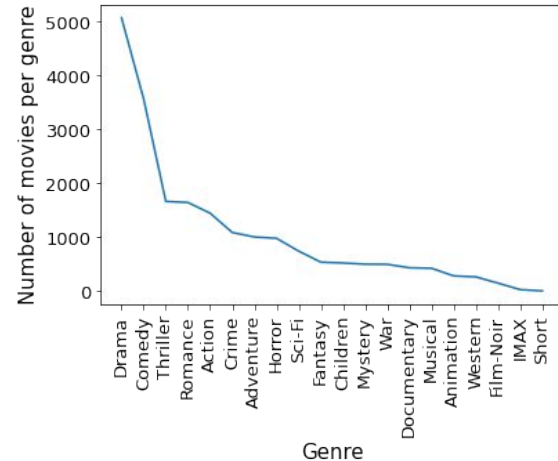
```
genre_dist = df_movies_genres['genrename']
.value_counts()
```

Drama	5076
Comedy	3566
Thriller	1664
Romance	1644
Action	1445

- **Simple line plot**

- `matplotlib.plot(genre_dist)`
- Set `x_label`, `y_label`, `title`, `font_sizes`, etc.
- Save figure as pdf, png, etc.

Genre distribution

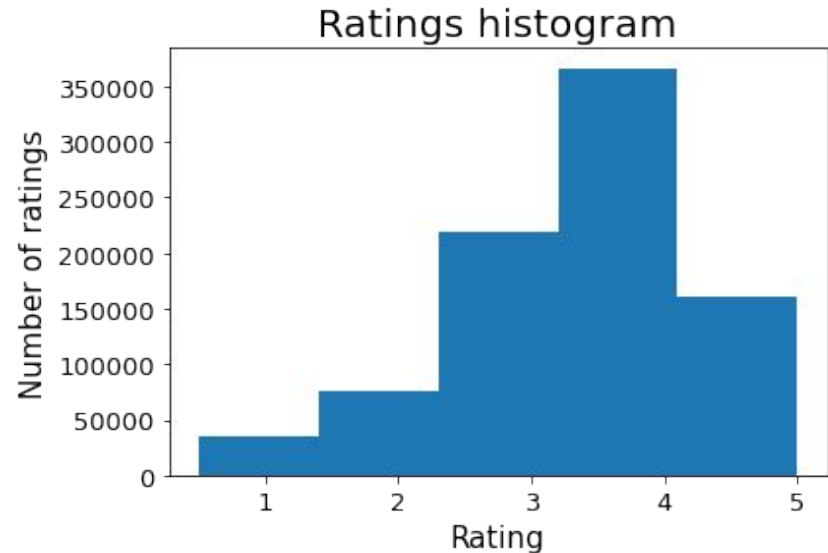


Matplotlib - Examples

- **Simple histogram**
 - **Setting**
 - `df_ratings` (SELECT * FROM usermovierating;)
 - `matplotlib.hist(df_ratings['rating'], bins=5)`
 - Set `x_label`, `y_label`, `title`, `font_sizes`, etc.
 - Save figure as pdf, png, etc.

Matplotlib - Examples

- **Simple histogram**
 - **Setting**
 - `df_ratings` (SELECT * FROM usermovierating;)
 - `matplotlib.hist(df_ratings['rating'], bins=5)`
 - Set `x_label`, `y_label`, `title`, `font_sizes`, etc.
 - Save figure as pdf, png, etc.
- **More positive than negative ratings**



Matplotlib - Examples

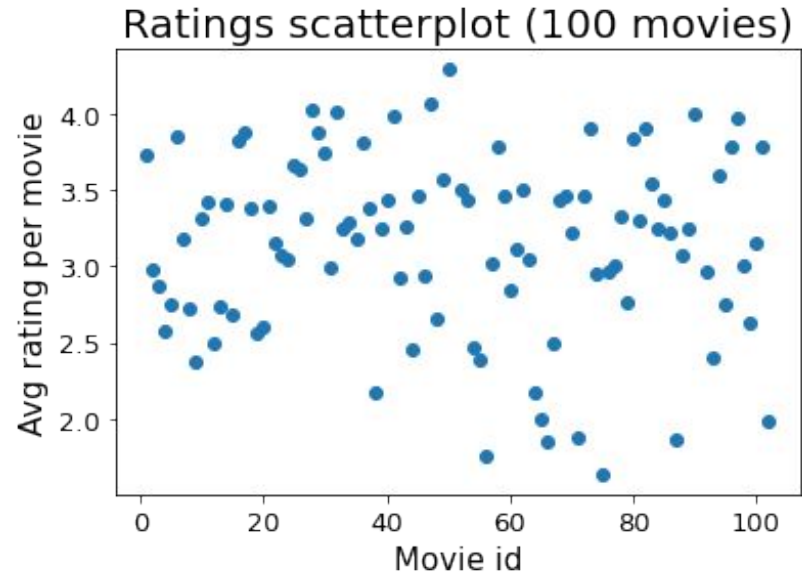
- **Scatter plot**
 - Plot average rating per movie

Matplotlib - Examples

- **Scatter plot**
 - Plot average rating per movie
 - `df_ratings.groupby('movieid')`
 - `matplotlib.scatter(x_values, y_values)`
 - X_values → movie-ids
 - Y_values → avg. ratings

Matplotlib - Examples

- **Scatter plot**
 - Plot average rating per movie
 - `df_ratings.groupby('movieid')`
 - `matplotlib.scatter(x_values, y_values)`
 - X_values → movie-ids
 - Y_values → avg. ratings
 - More positive avg. ratings than negative avg. ratings



Matplotlib - Examples

- **Scatter plot**
 - Plot average rating per movie
 - `df_ratings.groupby('movieid')`
 - `matplotlib.scatter(x_values, y_values)`
 - X_values → movie-ids
 - Y_values → avg. ratings
 - More positive avg. ratings than negative avg. ratings



Code Example

Questions / comments?

Do not forget to close DB connection at the end ;)

Group Project Presentations

Presentations

- **Mo. 12th & Mo. 19th of January 2026**
 - Probably 10 groups will be chosen to present on each date
 - Diversity of topics, if we have questions, etc.
 - Will let you know via TeachCenter / Discord

Presentations

- **Mo. 12th & Mo. 19th of January 2026**
 - Probably 10 groups will be chosen to present on each date
 - Diversity of topics, if we have questions, etc.
 - Will let you know via TeachCenter / Discord
- **You just use the slides that you have submitted**
 - Not the whole group needs to be present, although this would be nice
 - You can split up the presentation as you wish

Presentations

- **Mo. 12th & Mo. 19th of January 2026**
 - Probably 10 groups will be chosen to present on each date
 - Diversity of topics, if we have questions, etc.
 - Will let you know via TeachCenter / Discord
- **You just use the slides that you have submitted**
 - Not the whole group needs to be present, although this would be nice
 - You can split up the presentation as you wish
- **9 minutes per group**
 - 6 – 7 minutes presentation
 - Approx 2 minutes for questions
 - 1 minute to move to next group (plug in your laptop)

Conclusions

- **Summary**

- APIs
- Data handling and manipulation (Python, Pandas)
- DataFrames as a representation of a database table / result of a query
- Merging, analyzing, visualizing data

Conclusions

- **Summary**
 - APIs
 - Data handling and manipulation (Python, Pandas)
 - DataFrames as a representation of a database table / result of a query
 - Merging, analyzing, visualizing data

- **Next week + 15th of December**
 - No lectures (time to work on group project + prepare presentations)

- **Submit Exercise 2 and final group project slides (file names!)**

Conclusions

- **Summary**
 - APIs
 - Data handling and manipulation (Python, Pandas)
 - DataFrames as a representation of a database table / result of a query
 - Merging, analyzing, visualizing data
- **Next week + 15th of December**
 - No lectures (time to work on group project + prepare presentations)
- **Submit Exercise 2 and final group project slides (file names!)**

Course evaluation in TUG-Online → please fill it

A festive holiday scene featuring several lit candles in various holders (copper, silver, and blue), star-shaped ornaments, and string lights on a dark blue, textured background. The scene is decorated with pine branches and a string of small white stars. The overall atmosphere is warm and celebratory.

MERRY
CHRISTMAS
AND HAPPY NEW YEAR!