

# Pandas

# list and dict are not ideal for data analysis

```
# print students table
results = cursor.execute("SELECT * FROM students").fetchall()
for row in results:
    print(row)

# filter by age
for row in results:
    if row[3] > 25: # row[3] is age column
        print(row)

# calculate average age
total_age = 0
for row in results:
    total_age += row[3]
average_age = total_age / len(results)
```

# Pandas library for tabular data

```
import pandas as pd
```

---

<https://pandas.pydata.org>

# Pandas DataFrame

The diagram illustrates a Pandas DataFrame with the following structure and annotations:

- Column names:** Name, Team, Number, Position, Age, Height, Weight, College, Salary.
- Columns axis=1:** Points to the column headers.
- Index label:** Points to the index values (0-6).
- Index axis=0:** Points to the index values.
- Missing value:** Points to the 'NaN' value in the 'Number' column for index 3.
- Data:** Points to the numerical values in the 'Age', 'Height', 'Weight', and 'Salary' columns for index 4.

	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 Uniersity	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

# Pandas Series

Series		Series		DataFrame		
apples		oranges				
0	3	0	0	0	3	0
1	2	1	3	1	2	3
2	0	2	7	2	0	7
3	1	3	2	3	1	2

# Create a DataFrame

```
data = {  
    'name': ['John', 'Jane', 'Mary'],  
    'age': [25, 30, 27]  
}  
df = pd.DataFrame(data)
```

# DataFrame Index

## Get index of DataFrame

```
df.index
```

## Setting index for DataFrame

```
df.index = df['id']  
df = df.set_index('id')
```

# DataFrame Column

## Get column of DataFrame

```
df.columns
```

## Renaming columns

```
# for entire DataFrame  
df.columns = ['name', 'age', 'house_id']  
# for specific columns  
df.rename(columns={'name': 'full_name'})
```

## Deleting columns

```
df.drop(columns=['name'])
```



# Create DataFrame from database

## Option 1. Extract using `fetchall()`

```
results = conn.execute("SELECT * FROM students").fetchall() # list of tuples  
df = pd.DataFrame(results) # no column names
```

## Option 2. Extract using `read_sql()`

```
df = pd.read_sql("SELECT * FROM students", conn)
```

# SQL vs. Pandas

- Inspection
- Selection
- Filtering
- Sorting
- Aggregation
- Grouping
- Joining

## Read `harrypotter.db` into Pandas DataFrame

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('harrypotter.db')
df = pd.read_sql("SELECT * FROM students", conn)
```

# Inspection

## SQL

```
DESCRIBE students  
SELECT * FROM students LIMIT 5
```

## Pandas

```
df.info()  
df.describe()  
df.head()  
df.tail(3)
```

# Selection

## SQL

```
SELECT first_name FROM students;  
SELECT first_name, last_name FROM students;  
SELECT * FROM students;
```

## Pandas

```
df['first_name']  
  
cols = ['first_name', 'last_name']  
df[cols]  
df[['first_name', 'last_name']]  
df['first_name', 'last_name'] # Error  
  
df
```

## Create new columns

```
# vectorized operations
df['two'] = 2
df['age'] = 1997 - df['birthyear']
df['age2'] = df['age'] * 2
df['age3'] = df['age'] + df['age2']
```

# Filtering

- `query()` : **SQL-like syntax**
- `loc[]` : label-based
- `iloc[]` : position-based`
- ...

# Filtering

## SQL

```
SELECT * FROM students WHERE age = 10;  
SELECT first_name, house FROM students WHERE age > 10;  
SELECT * FROM students WHERE age in (10, 11);
```

## Pandas - `query`

```
df.query('age == 10')  
df.query('age > 10')[['first_name', 'house']]  
df.query('age in (10, 11)')
```



# Pattern matching

## SQL

```
SELECT * FROM students WHERE first_name LIKE 'J%';  
SELECT * FROM students WHERE first_name LIKE '%J';  
SELECT * FROM students WHERE first_name LIKE '%a%';
```

## Pandas - `query`

```
df.query("first_name.str.startswith('J')")  
df.query("first_name.str.endswith('J')")  
df.query("first_name.str.contains('a')")  
df.query("first_name.str.contains('a', case=False)")
```

# Sorting

## SQL

```
SELECT * FROM students ORDER BY age;  
SELECT * FROM students ORDER BY age desc;  
SELECT * FROM students ORDER BY age, first_name;
```

## Pandas

```
df.sort_values(by='age')  
df.sort_values(by='age', ascending=False)  
df.sort_values(by=['age', 'first_name'])
```

## Query `harrypotter.db` with Pandas

Find the answers to the following questions using Pandas `query()` function.

```
df = pd.read_sql("SELECT * FROM students", conn)
```

- What year was Harry Potter born?
- List the name of students who are born after 1980
- What is the name of the youngest student?

# Aggregation

```
df.agg({'column_name': 'function_name'})
```

# Aggregation

## SQL

```
SELECT AVG(age) FROM students;  
SELECT AVG(age), MAX(age) FROM students;
```

## Pandas

```
df.agg({'age': 'mean'})  
df.agg({'age': ['mean', 'max']})
```

---

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.aggregate.html>

# Grouping

```
df.groupby('column_name').agg({'column_name': 'function_name'})
```

# Grouping

## SQL

```
SELECT house_id, AVG(age) FROM students GROUP BY house_id;  
SELECT house_id, AVG(age), MAX(age) FROM students GROUP BY house_id;
```

## Pandas

```
df.groupby('house_id').agg({'age': 'mean'})  
df.groupby('house_id').agg({'age': ['mean', 'max']})
```

---

<https://realpython.com/pandas-groupby/>

# Joining

```
pd.merge(df1, df2, left_on='column_name', right_on='column_name')  
df1.merge(df2, left_on='column_name', right_on='column_name')
```



# Joining

## SQL

```
SELECT * FROM students JOIN houses ON students.house_id = houses.id;
```

## Pandas

```
# Join on column (default inner join)  
pd.merge(students, houses, left_on='house_id', right_on='id', how='inner')  
pd.merge(students, houses, left_on='house_id', right_on='id')
```

## SQL Murder Mystery

- Use SQL only to fetch the relevant tables
- Use Pandas `query()` to filter the records to get the same output as the SQL statement in each question.



## SQL Murder Mystery - Question 1

```
select id
from drivers_license
where hair_color = "red"
and car_make = "Tesla"
and car_model like "%Model S%"
and height between 65 and 67
```



## SQL Murder Mystery - Question 2

```
select *  
from facebook_event_checkin  
where event_name like "%SQL Symphony Concert%"  
and date between 20170101 and 20171231  
group by person_id
```

## SQL Murder Mystery - Question 3

**Hint: Use f-string and `list(id_list)` from Question 1**

```
select *
from person
where license_id in (
    select id
    from drivers_license
    where hair_color = "red"
    and car_make = "Tesla"
    and car_model like "%Model S%"
    and height between 65 and 67
)
```



## SQL Murder Mystery - Question 4

```
select *  
from person  
join drivers_license on person.license_id = drivers_license.id  
where hair_color = "red"  
and car_make = "Tesla"  
and car_model = "Model S"  
and height between 65 and 67
```

# Choosing between SQL and Pandas

## SQL:

- If doing a task in SQL can cut the amount of data returned to the client (e.g. by filtering)
- Data extraction, filtering, simple data analysis

## Pandas:

- If the amount of data returned to the client remains unchanged or grows by doing it in SQL (e.g. adding columns)
- Complex data analysis, formatting, etc.

**If it's painful or ugly, do it in Pandas**

	SQL	Pandas
Selection	<code>select name, age from students</code>	<code>df[['name', 'age']]</code>
Filtering	<code>select * from students where age &gt; 10</code>	<code>df.query('age &gt; 10')</code>
Sorting	<code>select * from students order by age</code>	<code>df.sort_values(by = 'age')</code>



	SQL	Pandas
Aggregation	<pre>SELECT AVG(age) FROM students</pre>	<pre>df.agg({'age': 'mean'})</pre>
Grouping	<pre>SELECT house, AVG(age), MAX(age) FROM students GROUP BY house</pre>	<pre>df.groupby('house').agg({'age': ['mean', 'max']})</pre>
Joining	<pre>SELECT * from students join houses on students.house_id = houses.id</pre>	<pre>pd.merge(df, df2, left_on = 'house_id', right_on = 'id')</pre>

# References

- <https://www.datacamp.com/tutorial/pandas>
- [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/10min.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html)