Pandas

`list` and `dict` are not suitable for data analysis

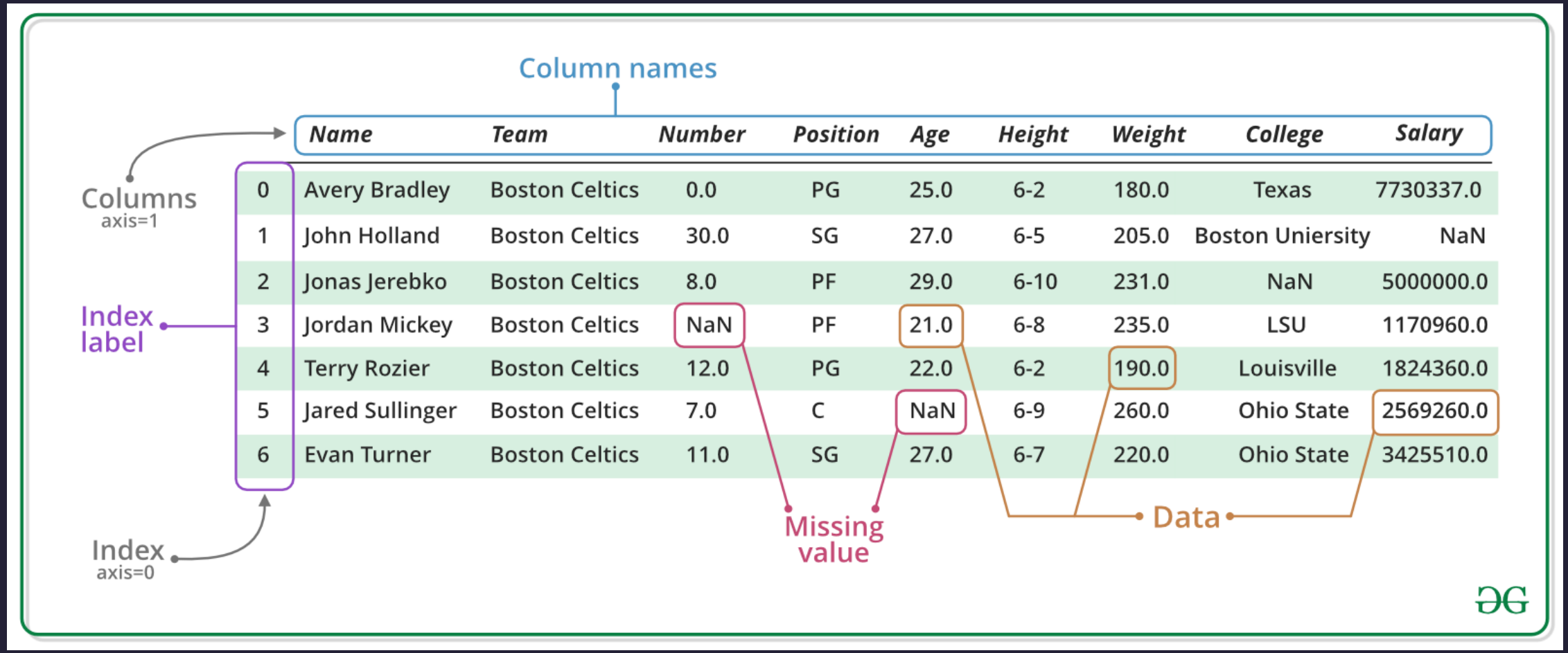
```
# print students table
results = cursor.execute("SELECT * FROM students").fetchall()
for row in results:
    print(row)

# 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

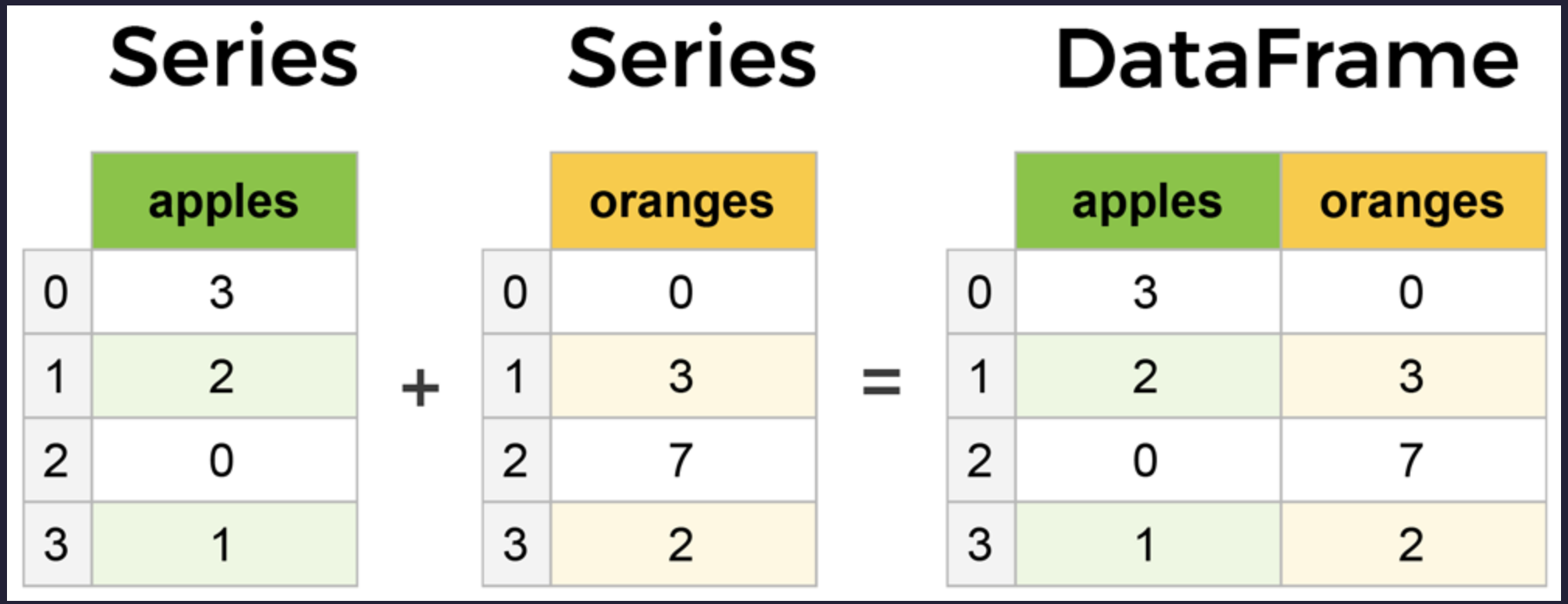
```
import pandas as pd
```

https://pandas.pydata.org

Pandas DataFrame



Pandas Series



Create a DataFrame

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

More on DF creation

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

list_of_list = [    # list of tuples also works
    ['John', 25],
    ['Jane', 30],
    ['Mary', 27]
]
df = pd.DataFrame(list_of_list, columns=['name', 'age'])

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

Index

Get index of DataFrame

```
df.index
```

Setting index for DataFrame

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

Column

Get column of DataFrame

```
df.columns
```

Deleting columns

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

Renaming columns

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

Data import & export in Pandas

```
# Import from CSV
df = pd.read_csv('students.csv')

# Export to CSV
df.to_csv('students.csv')

# Import from Excel
df = pd.read_excel('students.xlsx')

# Export to Excel
df.to_excel('students.xlsx')
```

Importing from database

Connect to database

```
conn = sqlite3.connect('harrypoter.db')
```

Option 1. Extract using `fetchall()` (no column names)

```
results = conn.execute("SELECT * FROM students").fetchall()
df = pd.DataFrame(results)
```

Option 2. Extract using `read_sql()`

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

Create new columns

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

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

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

Pandas

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

Selection

SQL

```sql
SELECT first_name FROM students;
SELECT first_name, age FROM students;
SELECT * FROM students;
```

Pandas

```
df['first_name']
df[['first_name', 'age']]
df
```

Filtering

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

Filtering - `query`

SQL

```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`

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

Pattern matching - `query`

SQL

```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`

```python
# Using query()
df.query("first_name.str.startswith('J')")
df.query("first_name.str.endswith('J')")
df.query("first_name.str.contains('a')")
```

Multiline query

```
query = """
    age > 10 \
    and first_name.str.startswith('J') \
    and first_name.str.endswith('J') \
    and first_name.str.contains('a')
"""

df.query(query)
```

Sorting

SQL

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

Pandas

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

Chaining methods

```
df.query('age > 10').sort_values(by='age')

df.query('age > 10').sort_values(by='age').head(5)

(
    df
    .query('age > 10')
    .sort_values(by='age')
    .head(5)
    .tail(3)
)
```

💻 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 oldest student?

1. What year was Harry Potter born?

```
SELECT * FROM students WHERE first_name = "Harry";
```

```
df.query('first_name == "Harry"')
```

2. List the name of students who are born after 1980

```
SELECT first_name, last_name, birthyear FROM students WHERE birthyear = 1980;
```

```
df.query('birthyear == 1980')[['first_name', 'last_name', 'birthyear']]
```

3. What is the name of the oldest student?

```
SELECT * FROM students ORDER BY birthyear DESC LIMIT 1;
```

```
df.sort_values(by='birthyear', ascending=False).head(5)
```

Aggregation

SQL

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

Pandas

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

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

Grouping

SQL

```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

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

---

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

Joining

SQL

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

Pandas

```python
# Join on column
pd.merge(students, houses, left_on='house_id', right_on='id', how='inner')
pd.merge(students, houses, left_on='house_id', right_on='id')
# Join on index
pd.merge(students, houses, left_index=True, right_index=True, how='inner')
pd.merge(students, houses, left_index=True, right_index=True)
```

💻 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
```

```
drivers_license = pd.read_sql("select * from drivers_license", conn)
query = """
    hair_color == "red" \
    and car_make == "Tesla" \
    and car_model.str.contains("Model S") \
    and height >= 65 and height <= 67 \
"""
id_list = drivers_license.query(query)['id']
```

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
```

```
facebook_event_checkin = pd.read_sql("select * from facebook_event_checkin", conn)
query = """
    event_name == "SQL Symphony Concert" \
    and date >= 20170101 and date <= 20171231
"""
(
    facebook_event_checkin
    .query(query)
    .groupby('person_id')
    .agg({'date': 'count'})
)
```

💻 SQL Murder Mystery - Question 3

Hint: Use f-string

```
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
)
```

```
person.query(f'license_id in {id_list.to_list()}')
```

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
```

```
query = """
    hair_color == "red" \
    and car_make == "Tesla" \
    and car_model.str.contains("Model S") \
    and height >= 65 and height <= 67 \
"""
(
    person
    .merge(drivers_license, how='inner', left_on = 'license_id', right_on = 'id')
    .query(query)
)
```

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.

---

https://towardsdatascience.com/sql-vs-pandas-how-to-balance-tasks-between-server-and-client-side-9e2f6c95677

	SQL	Pandas
Selection	`select name, age from students`	`df[['name', 'age']]`
Filtering	`select * from students where age > 10`	`df.query('age > 10')` `df.loc[df['age'] > 10]`
Sorting	`select * from students order by age`	`df.sort_values(by = 'age')`

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

References

- https://www.datacamp.com/tutorial/pandas
- https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html