Relational data

- SQLite
- pandas

Two tables

Table: country

name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

Table: city

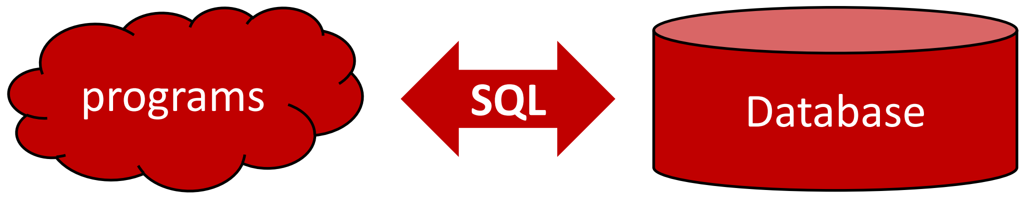
name	country	population	established
'Copenhagen'	'Denmark'	775033	800
'Aarhus'	'Denmark'	273077	750
'Berlin'	'Germany'	3711930	1237
'Munich'	'Germany'	1464301	1158
'Reykjavik'	'Iceland'	126100	874
'Washington D.C.'	'USA'	693972	1790
'New Orleans'	'USA'	343829	1718
'San Francisco'	'USA'	884363	1776

SQL

pronounced ˌɛsˌkjuːˈɛl or ˈsiːkwəl

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

- SQL = Structured Query **Language**
- **Database = collection of tables** stored persistently on disk
- ANSI and ISO standards since 1986 and 1987, respectively; origin early 70s
- Widespread used SQL databases (can handle many tables/rows/users): Oracle, MySQL, Microsoft SQL Server, PostgreSQL and IBM DB2
- **SQLite** is a very lightweight version storing a database in a **single file**, without a separate database server; first release in 2000
- SQLite is included in both iOS and Android mobil phones and operating systemes like Mac OS, Windows, and Fedora Linux



The Course "Databases" gives a more in-depth introduction to SQL (MySQL)

Examples : SQL statements

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

- CREATE TABLE country (name, population, area, capital)
- INSERT INTO country VALUES ('Denmark', 5748769, 42931, 'Copenhagen')
- UPDATE country SET population=5748770 WHERE name='Denmark'
- SELECT name, capital FROM country WHERE population >= 1000000
  > [('Denmark', 'Copenhagen'), ('Germany', 'Berlin'), ('USA', 'Washington, D.C.')]
- SELECT * FROM country WHERE capital = 'Berlin'
  > [('Germany', 82800000, 357168, 'Berlin')]
- SELECT country.name, city.name, city.established FROM city, country WHERE city.name=country.capital AND city.population < 700000
  > [('Iceland', 'Reykjavik', 874), ('USA', 'Washington, D.C.', 1790)]
- DELETE FROM country WHERE name = 'Germany'
- DROP TABLE country

www.w3schools.com/sql/

SQLite

sqlite-example.py

```python
import sqlite3
connection = sqlite3.connect('example.sqlite')  # creates file if necessary
c = connection.cursor()
c.executescript('''DROP TABLE IF EXISTS country;  -- multiple SQL statements
                   DROP TABLE IF EXISTS city''')
countries = [('Denmark', 5748769, 42931, 'Copenhagen'),
             ('Germany', 82800000, 357168, 'Berlin'),
             ('USA', 325719178, 9833520, 'Washington, D.C.'),
             ('Iceland', 334252, 102775, 'Reykjavik')]
cities = [('Copenhagen', 'Denmark', 775033, 800),
          ('Aarhus', 'Denmark', 273077, 750),
          ('Berlin', 'Germany', 3711930, 1237),
          ('Munich', 'Germany', 1464301, 1158),
          ('Reykjavik', 'Iceland', 126100, 874),
          ('Washington, D.C.', 'USA', 693972, 1790),
          ('New Orleans', 'USA', 343829, 1718),
          ('San Francisco', 'USA', 884363, 1776)]
c.execute('CREATE TABLE country (name, population, area, capital)')
c.execute('CREATE TABLE city (name, country, population, established)')
c.executemany('INSERT INTO country VALUES (?,?,?,?)', countries)
c.executemany('INSERT INTO city VALUES (?,?,?,?)', cities)
connection.commit()  # save data to database before closing
connection.close()
```

docs.python.org/3/library/sqlite3.html

SQLite query examples

try to avoid using the asterisk (*) as a good habit
www.sqlitetutorial.net/sqlite-select

sqlite-example.py

```
for row in c.execute('SELECT * FROM country'):  # * = all columns, execute returns iterator
    print(row)                                  # row is by default a Python tuple

for row in c.execute('''SELECT * FROM city, country    -- all pairs of rows from city × country
                        WHERE city.name = country.capital AND city.population < 700000'''):
    print(row)

print(*c.execute('''SELECT country.name,
                           COUNT(city.name) AS cities,
                           100 * SUM(city.population) / country.population
                    FROM city JOIN country ON city.country = country.name  -- SQL join 2 tables
                    WHERE city.population > 500000                      -- only consider big cities
                    GROUP BY city.country                           -- output has one row per group of rows
                    ORDER BY cities DESC, SUM(city.population) DESC'''))   # ordering of output
```

Python shell

```
| ('Denmark', 5748769, 42931, 'Copenhagen')
| ('Germany', 82800000, 357168, 'Berlin')
| ('USA', 325719178, 9833520, 'Washington, D.C.')
| ('Iceland', 334252, 102775, 'Reykjavik')

| ('Reykjavik', 'Iceland', 126100, 874, 'Iceland', 334252, 102775, 'Reykjavik')
| ('Washington, D.C.', 'USA', 693972, 1790, 'USA', 325719178, 9833520, 'Washington, D.C.')

| ('Germany', 2, 6) ('USA', 2, 0) ('Denmark', 1, 13)
```

SQL injection

Right way

```
c.execute('INSERT INTO users VALUES (?)', (user,))
```

can execute a string containing several SQL statements

unsafe-example.py

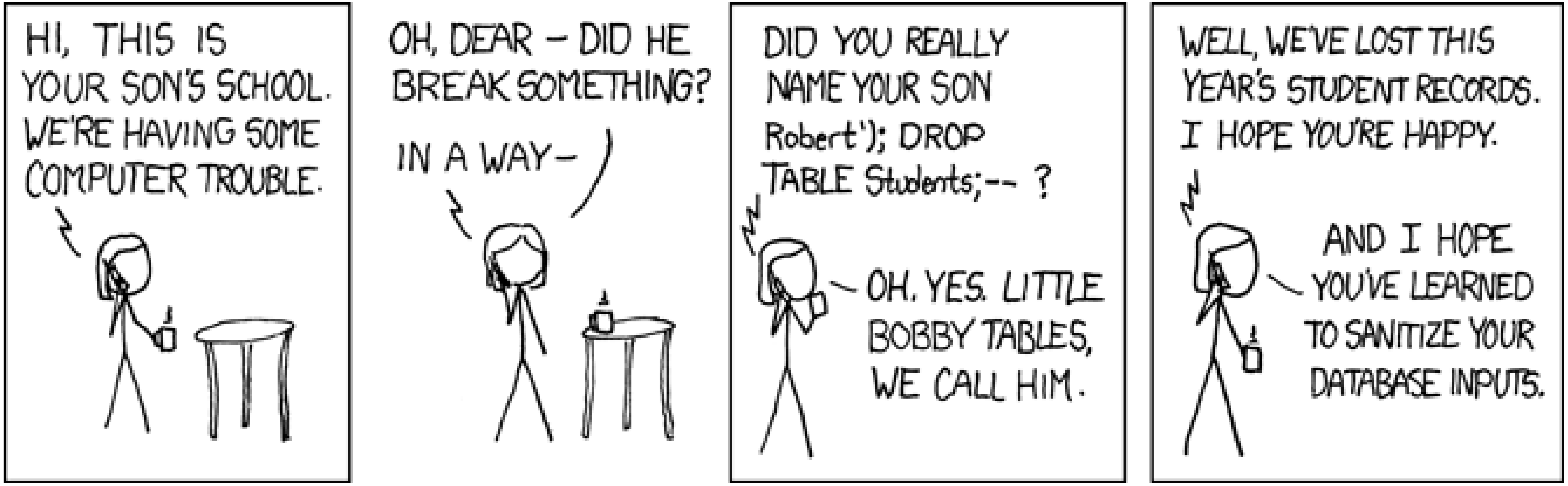
```
import sqlite3
connection = sqlite3.connect('users.sqlite')
c = connection.cursor()
c.execute('CREATE TABLE users (name)')
while True:
    user = input('New user: ')
    c.executescript(f'INSERT INTO users VALUES ("{user}")')
    connection.commit()
    print(list(c.execute('SELECT * FROM users')))
```

Insecure
NEVER use f-string with user input

Python shell

```
> New user: gerth
| [('gerth',)]
> New user: guido
| [('gerth',), ('guido',)]
> New user: evil"); DROP TABLE users; --
| sqlite3.OperationalError: no such table: users
```

```
INSERT INTO users VALUES ("evil"); DROP TABLE users; --")
```


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

xkcd.com/327/

Pandas

- Comprehensive Python library for data manipulation and analysis, in particular tables and time series
- Pandas **data frames** = tables
- Supports interaction with SQL, CSV, JSON, ...
- Integrates with Jupyter, numpy, matplotlib, ...

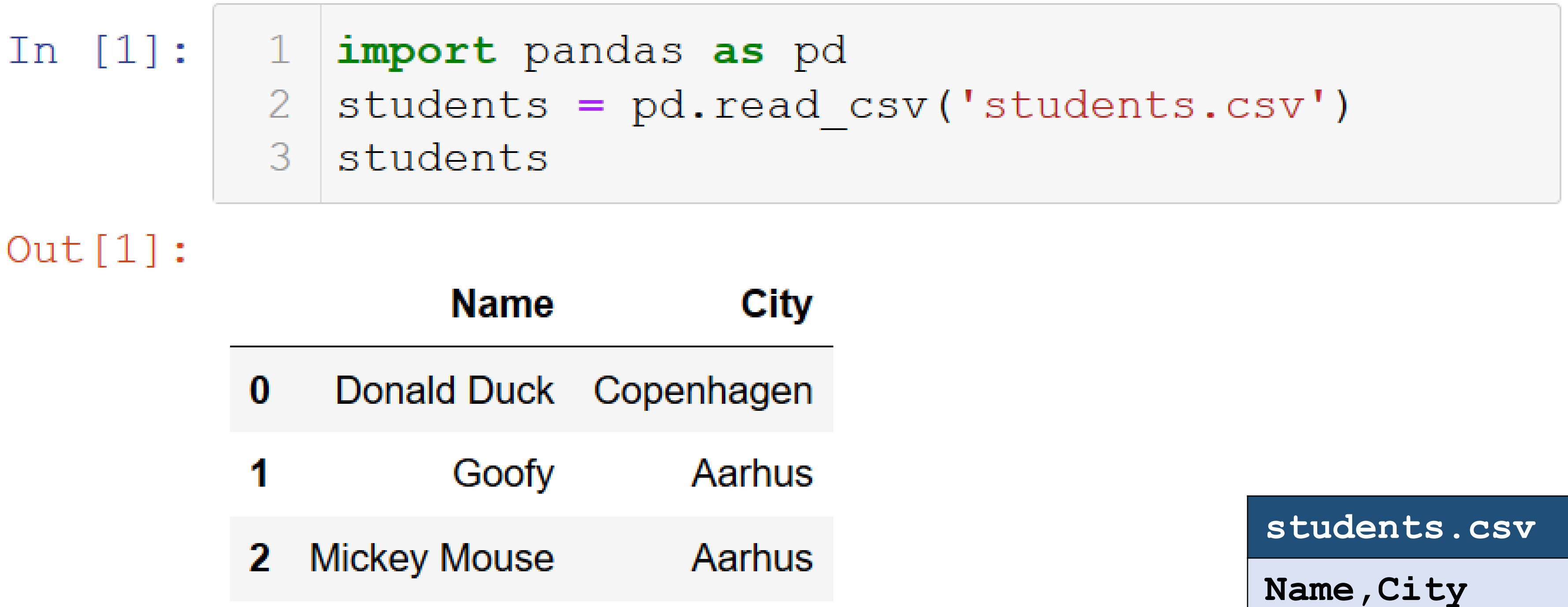
pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

pandas.pydata.org

Pandas integration with Jupyter

- Tables (Pandas data frames) are rendered nicely in Jupyter



In [1]:
```python
import pandas as pd
students = pd.read_csv('students.csv')
students
```

Out[1]:

	Name	City
0	Donald Duck	Copenhagen
1	Goofy	Aarhus
2	Mickey Mouse	Aarhus

students.csv

```
Name,City
"Donald Duck","Copenhagen"
"Goofy","Aarhus"
"Mickey Mouse","Aarhus”
```

Reading tables (data frames)

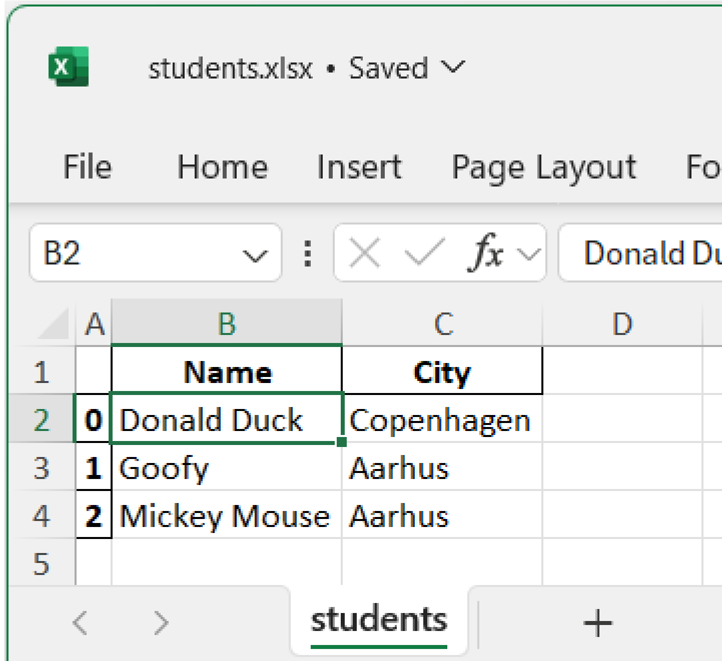
- Pandas provides functions for reading and writting pandas.DataFrames as files in different data formats, e.g. SQLite, csv and EXCEL files

pandas-example.py

```
import pandas as pd
import sqlite3
connection = sqlite3.connect('example.sqlite')
countries = pd.read_sql_query('SELECT * FROM country', connection)
cities = pd.read_sql_query('SELECT * FROM city', connection)
students = pd.read_csv('students.csv')
students.to_sql('students', connection, if_exists='replace')
students.to_excel('students.xlsx', sheet_name='students')
print(students)
```

Python shell

```
|              Name        City
| 0    Donald Duck  Copenhagen
| 1          Goofy      Aarhus
| 2   Mickey Mouse      Aarhus
```



	A	B	C	D
1		Name	City	
2	0	Donald Duck	Copenhagen	
3	1	Goofy	Aarhus	
4	2	Mickey Mouse	Aarhus	
5				

Selecting columns and rows

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

Python shell

```
> countries['name']                  # select column
> countries.name                     # same as above
> countries[['name', 'capital']]     # select multiple columns, note double-[]
> countries.head(2)                  # first 2 rows
> countries[1:3]                     # slicing rows, rows 1 and 2
> countries[::2]                     # slicing rows, rows 0 and 2
> countries.at[1, 'area']            # indexing cell by (row label, column name)
> cities[(cities['name'] == 'Berlin') | (cities['name'] == 'Munich')]  # select rows
|      name   country  population  established
| 2  Berlin   Germany     3711930         1237  # note original row labels
| 3  Munich   Germany     1464301         1158
> pd.DataFrame([[1, 2], [3, 4], [5, 6]], columns=['x', 'y'])  # create DF from list
> pd.DataFrame(np.random.random((3, 2)), columns=['x', 'y'])  # from numpy
```

pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Row labels

Python shell

```
> df = pd.DataFrame(np.arange(1, 13).reshape(3, 4),
                    index=['q', 'w', 'e'],         # row labels
                    columns=['c', 'a', 'd', 'e'])  # column names
> df
|    c   a   d   e
| q  1   2   3   4   # row labels can be strings
| w  5   6   7   8
| e  9  10  11  12
> df.loc['w':'e', ['e', 'a']]  # slice of labeled rows
|     e   a
| w   8   6
| e  12  10
> df.loc['w']  # single row (a one-dimensional pd.Series)
| c     5
| a     6
| d     7
| e     8
| Name: w, dtype: int32
> df.iloc[:2,:2]  # use iloc to work with integer indexes
|    c  a
| q  1  2
| w  5  6
```

Merging tables and creating a new column

pandas-example.py

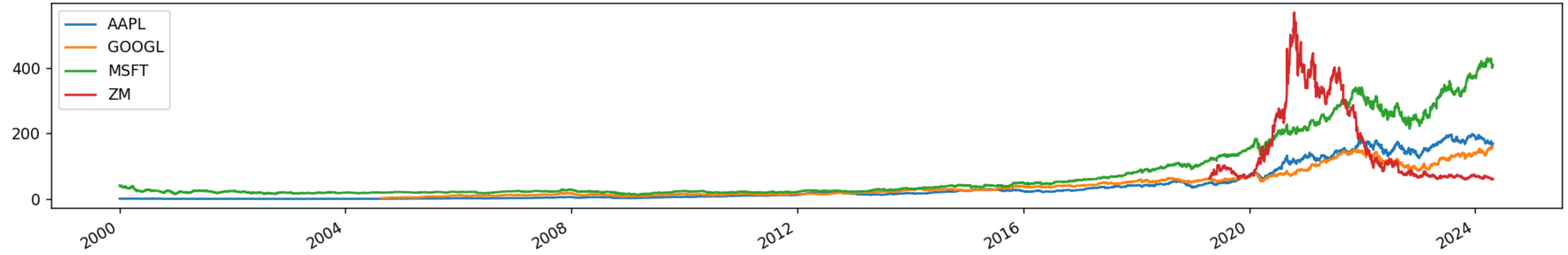
```
M = pd.merge(countries, cities, left_on='capital', right_on='name')
# both data frames had a 'name' and 'population' column
M1 = M.rename(columns={
    'population_x': 'country_population',
    'population_y': 'capital_population'
})
M2 = M1.drop(columns=['name_x', 'name_y'])
M2['%pop in capital'] = M2.capital_population / M2.country_population  # add column
M2.sort_values('%pop in capital', ascending=False, inplace=True)
print(M2[['country', '%pop in capital']])
```

Python shell

```
|    country  %pop in capital
| 3  Iceland         0.377260  # note row labels are permuted
| 0  Denmark         0.134817
| 1  Germany         0.044830
| 2      USA         0.002131
```

Pandas datareader and Matplotlib

- pandas_datareader provides access to many data sources
- dataframes have a .plot method (using matplotlib.pyplot)
- `pip install pandas-datareader` and `pip install setuptools`



pandas-datareader.py

```
import matplotlib.pyplot as plt
import pandas_datareader
#df = pandas_datareader.data.DataReader(['AAPL', 'GOOGL', 'MSFT', 'ZM'], 'stooq')  # ignores start=...
df = pandas_datareader.stooq.StooqDailyReader(['AAPL', 'GOOGL', 'MSFT', 'ZM'], start='2000-01-01').read()
df['Close'].plot()
plt.legend()
plt.show()
```

pandas-datareader.readthedocs.io
pandas-datareader.readthedocs.io/en/latest/readers/stooq.html

Hierarchical / Multi-level indexing (MultiIndex)

Python shell

```
> df.tail(2)
| Attributes    Close                       ...      Volume
| Symbols         AAPL     GOOGL     MSFT   ...       GOOGL        MSFT          ZM
| Date                                      ...
| 2020-04-29   287.73  1342.18   177.43   ...   5417888.0  51286559.0  22033320.0
| 2020-04-30   293.80  1346.70   179.21   ...   2788644.0  53627543.0  16648922.0
> df['Close'].tail(2)
| Symbols         AAPL     GOOGL     MSFT       ZM
| Date
| 2020-04-29   287.73  1342.18   177.43   146.48
| 2020-04-30   293.80  1346.70   179.21   135.17
> df['Close']['GOOGL'].tail(2)
| Date
| 2020-04-29    1342.18
| 2020-04-30    1346.70
| Name: GOOGL, dtype: float64
> df.loc[:, pd.IndexSlice[:,'GOOGL']].tail(2)
| Attributes      Close      High       Low      Open       Volume
| Symbols         GOOGL     GOOGL     GOOGL     GOOGL        GOOGL
| Date
| 2020-04-29   1342.18   1360.15   1326.73   1345.00   5417888.0
| 2020-04-30   1346.70   1350.00   1321.50   1331.36   2788644.0
```

Both rows and columns can have multi-level indexing

Python shell

```
> df.columns
| MultiIndex([( 'Close',  'AAPL'),
|             ( 'Close', 'GOOGL'),
|             ( 'Close',  'MSFT'),
|             ( 'Close',    'ZM'),
|             (  'High',  'AAPL'),
|             (  'High', 'GOOGL'),
|             (  'High',  'MSFT'),
|             (  'High',    'ZM'),
|             (   'Low',  'AAPL'),
|             (   'Low', 'GOOGL'),
|             (   'Low',  'MSFT'),
|             (   'Low',    'ZM'),
|             (  'Open',  'AAPL'),
|             (  'Open', 'GOOGL'),
|             (  'Open',  'MSFT'),
|             (  'Open',    'ZM'),
|             ('Volume',  'AAPL'),
|             ('Volume', 'GOOGL'),
|             ('Volume',  'MSFT'),
|             ('Volume',    'ZM')],
|   names=['Attributes', 'Symbols'])
```

pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html