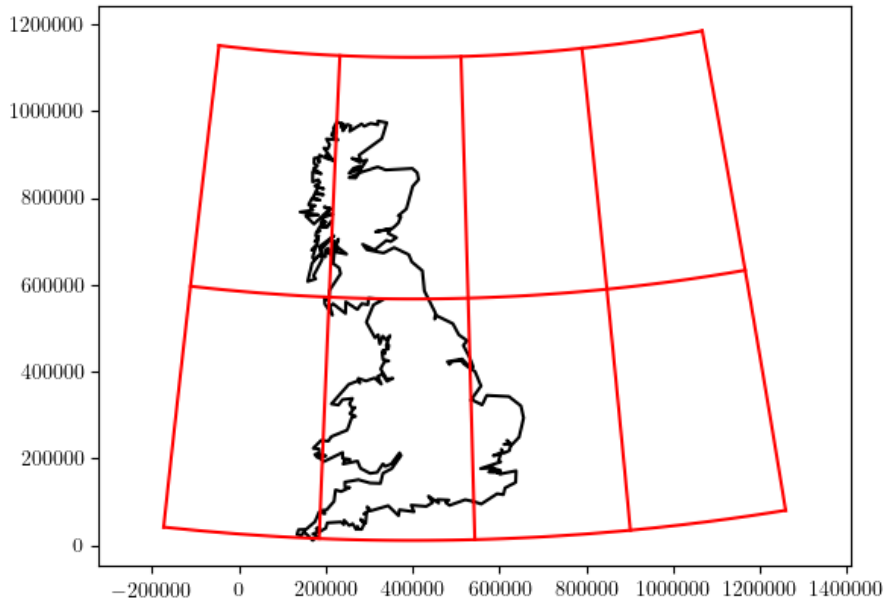
Information for the damage mapper/postcode locator

Thomas Davison

- thomas.davison@imperial.ac.uk
- RSM 4.85

A FEW NOTES ON (GEO)SPATIAL INDEXING

- Postcode data is linked to latitude and longitude.
- Pressure and damage at a given radius is given in metres.
- Not trivial to convert between these two.



A FEW NOTES ON (GEO)SPATIAL INDEXING

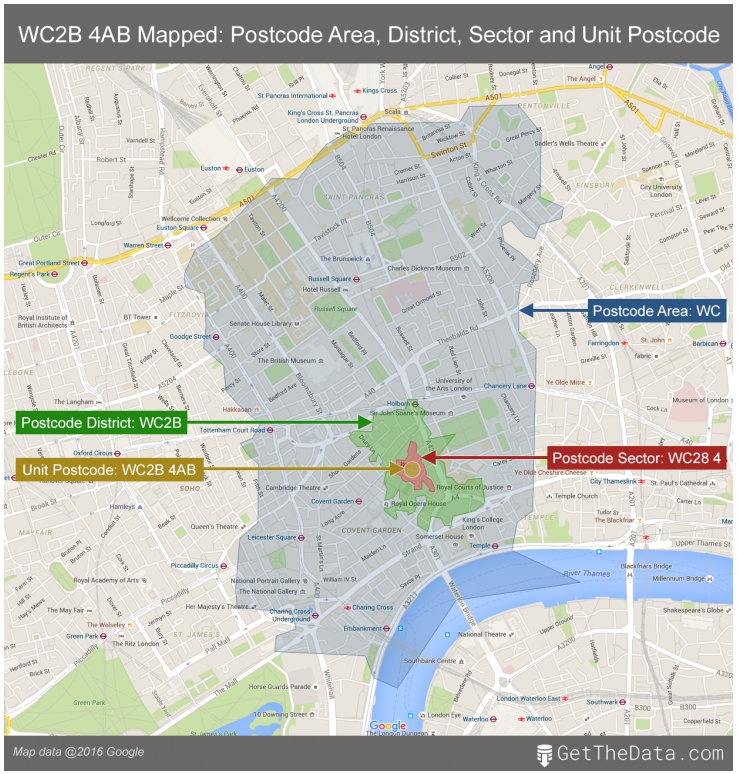
- Data has been given in latitude-longitude pairs
- You will need to find nearby points by surface distance in metres.
- Full conversion is non-trivial, but you should assume a **spherical, non-rotating** Earth
- OK to rearrange/preprocess data for faster lookups

A FEW NOTES ON (GEO)SPATIAL INDEXING

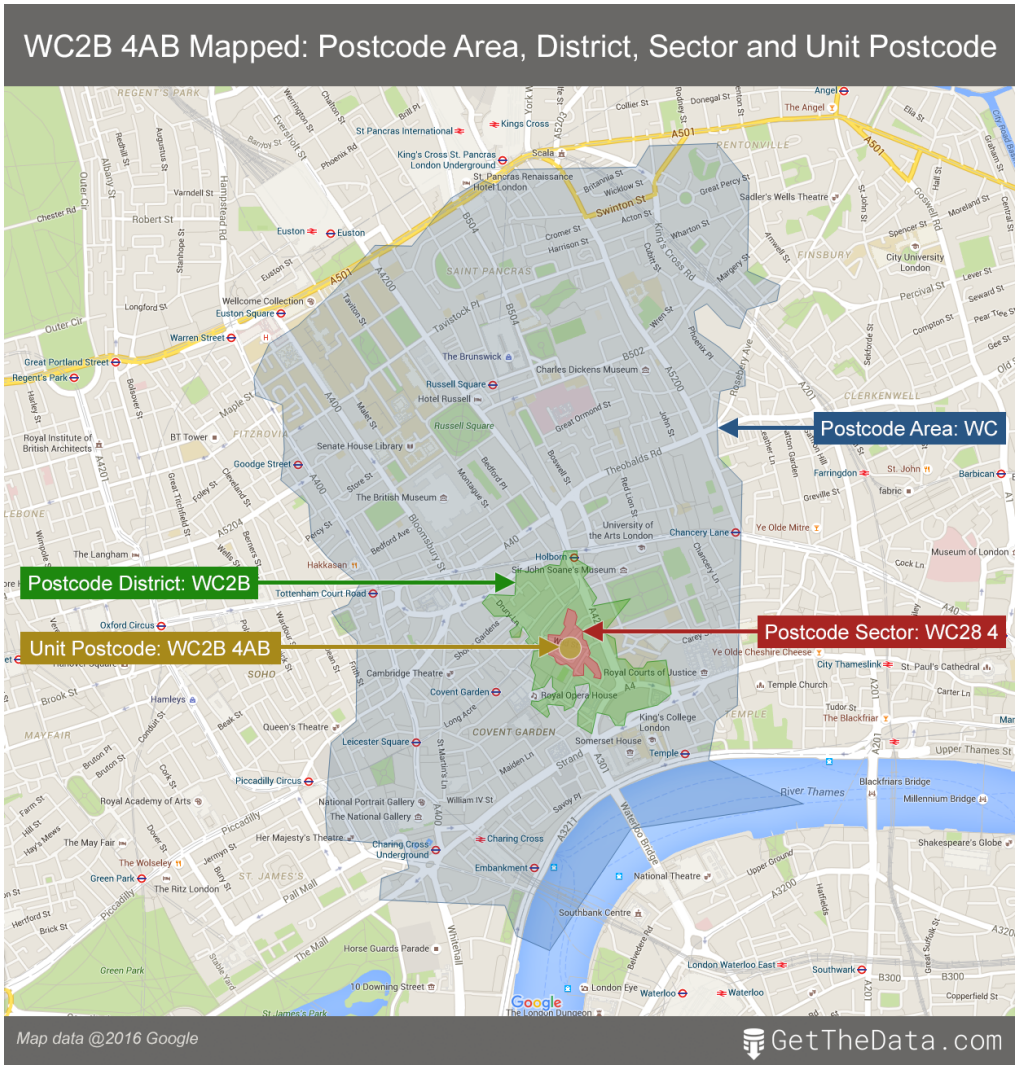
- Can be worth sieving data with a cheap check before doing expensive one.
- Many, many approaches (bounding boxes, gridding, R-trees, kd-trees, etc...)
- Given limited development time, start simple
- Remember that numpy and pandas calls can be 100s times faster than hand written Python.

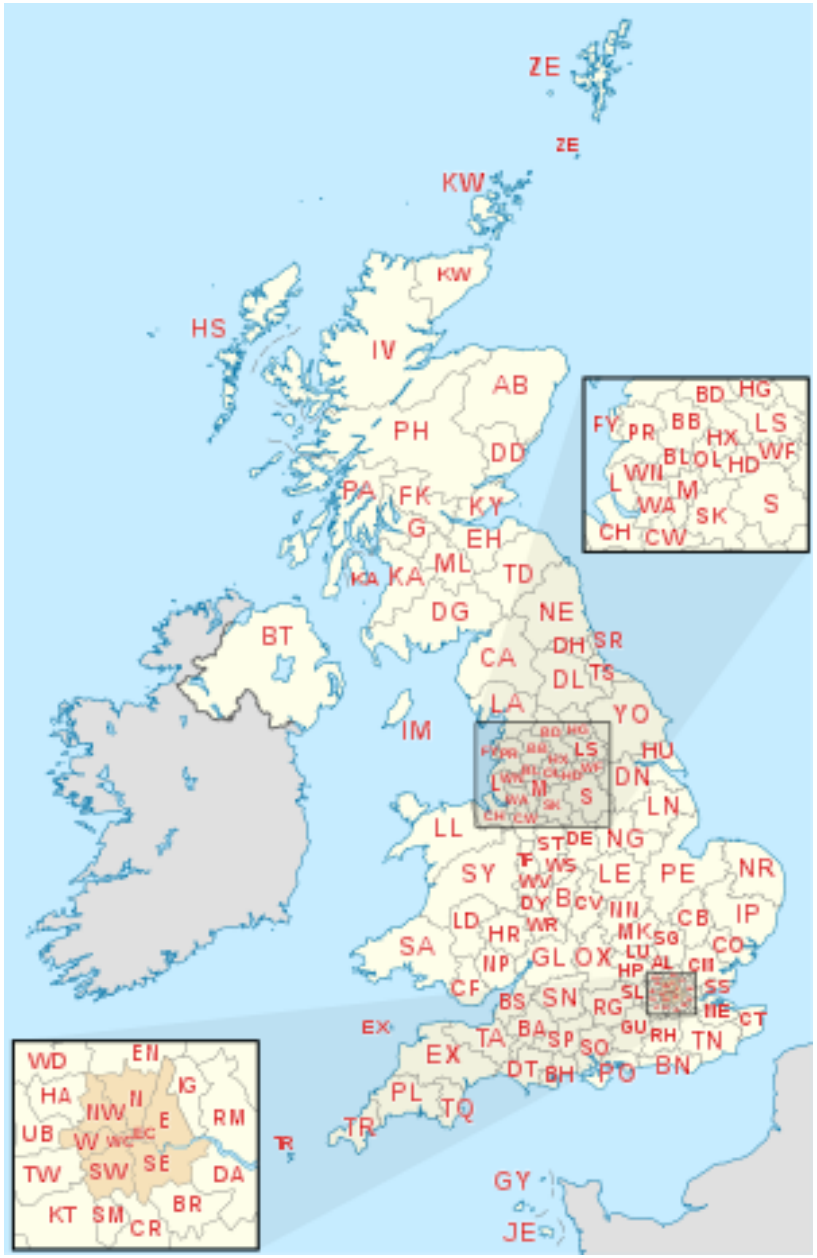
A FEW NOTES ON POSTCODES

- UK name for postal code, like US ZIP code, Chinese 邮政编码 etc.
- Divided into outward and inward sections
- Outward has area (eg. WC) & district (eg. 2B)
- Inward has sector (eg. 4) & unit (eg. AB)
- Outward 3 or 4 characters, inward 3



Postcode locations




ZE
ZE
KW
KW
HS
IV
AB
PH
DD
PA
FK
KY
G
EH
KA
KA
ML
TD
DG
NE
BT
DH
SR
CA
DL
TS
LA
YO
IM
HU
DN
LN
LL
ST
DE
NG
SY
WS
LE
PE
NR
DY
B
CV
IN
LD
WR
CB
IP
HR
MK
SG
SA
GL
OX
LU
CO
NP
HP
AL
CF
BS
SN
RG
SL
SS
EX
TA
BA
SP
GU
RH
TN
CT
EX
DT
BH
SO
PO
BN
PL
TQ
TR
GY
JE
BD
HG
FY
PR
BB
LS
HX
WF
BL
OL
HD
L
WN
M
WA
SK
S
CH
CW
WD
EN
HA
N
IG
NW
E
RM
UB
W
WC
EC
TW
SW
SE
DA
KT
SM
CR
BR

Many different postcode formats in use on computers

Single space:

- `SW19 5AE`
- `SW7 2AZ`

Fixed length, 7 characters. **We strongly recommend you use this format in your software**:

- `SW195AE`
- `SW7 2AZ`

Fixed length, 8 characters:

- `SW19 5AE`
- `SW7  2AZ`

TESTING

- Stub has limited pytest tests for existing code
- Tests are partly set up to check formatting will be scorable.
    - Keep existing tests. Add new tests as you add new features
- GitHub Actions is set up to run on all code pushed (using Python 3.10 on linux)
- Strongly recommended to carry on adding tests as you go. Quicker to stick to linux

PLANNING

- Final division of work is up to to you.
- Can be democratic, or work for a (sub)manager.
- Important to keep talking to each other.
- Keep an eye on the critical path.

Supplementary information for damage mapper tool

(`DamageMapper.ipynb`)

Airblast damage

The main cause of damage close to the impact site is a strong (pressure) blastwave in the air, known as the **airblast**. Empirical data suggest that the pressure in this wave p (in Pa) (above ambient, also known as overpressure), as a function of explosion energy E_k (in kilotons of TNT equivalent), burst altitude z_b (in m) and horizontal range r (in m), is given by:

$$p(r) = 3.14 \times 10^{11} \left(\frac{r^2 + z_b^2}{E_k^{2/3}} \right)^{-1.3} + 1.8 \times 10^{7} \left(\frac{r^2 + z_b^2}{E_k^{2/3}} \right)^{-0.565}$$

Geographic data files

The geographic data supplied by running the `download_data.py` script consists of two files.

The larger file is `full_postcodes.csv`, which contains:

- a list of current UK postcodes
- a government-assigned code designating the local administrative area
- information information on the average (mean) longitude and latitude of the addresses comprising the unit, using the international WGS 84 geodetic datum as supported by modern GPS.

In [2]:

```
import pandas as pd
postcodes = pd.read_csv('./resources/full_postcodes.csv')
postcodes.head()
```

Out[2]:

	Postcode	LAU218CD	Latitude	Longitude
0	AB101AB	S31000932	57.149606	-2.096916
1	AB101AF	S31000932	57.148707	-2.097806
2	AB101AG	S31000932	57.149051	-2.097004
3	AB101AH	S31000932	57.148080	-2.094664
4	AB101AL	S31000932	57.150058	-2.095916

Geographic data files

The smaller file is `population_by_postcode_sector.csv`, which contains 2011 census data arranged by postcode sector.

The important columns for this project are the postcode sector (`"geography"`) and the total population (`"All usual residents"`), although you are welcome to use other data in your tool if you wish.

In [3]:

```
census = pd.read_csv('./resources/population_by_postcode_sector.csv')
census.head()
```

Out[3]:

	date	geography	geography code	Rural Urban	Variable: All usual residents; measures: Value	Variable: Males; measures: Value	Variable: Females; measures: Value	Variable: Lives in a household; measures: Value	Variable: Lives in a communal establishment; measures: Value	Variable: Schoolchild or full-time student aged 4 and over at their non term-time address; measures: Value	Variabl Are (Hectares measure Valu
0	2011	AL1 1	AL1 1	Total	5453	2715	2738	5408	45	75	225.6
1	2011	AL1 2	AL1 2	Total	6523	3183	3340	6418	105	77	286.5
2	2011	AL1 3	AL1 3	Total	4179	2121	2058	4100	79	46	97.1
3	2011	AL1 4	AL1 4	Total	9799	4845	4954	9765	34	285	244.7
4	2011	AL1 5	AL1 5	Total	10226	5129	5097	10211	15	133	200.9

Notes on distance, bearing and position

To determine the surface zero location (the point on Earth's surface that is closest to the point of airburst) a useful set of spherical geometric formulae relate the bearing, β (also known as forward azimuth) to take to get from one point to another along a great circle,

$$\tan\beta = \frac{\cos\varphi_2 \sin(\lambda_2 - \lambda_1)}{\cos\varphi_1 \sin\varphi_2 - \sin\varphi_1 \cos\varphi_2 \cos(\lambda_2 - \lambda_1)},$$

where λ is longitude and φ is latitude.

The related problem of the final destination given a surface distance and initial bearing is given by:

$$\sin\varphi_2 = \sin\varphi_1 \cos\left(\frac{r}{R_p}\right) + \cos\varphi_1 \sin\left(\frac{r}{R_p}\right)\cos\beta,$$

$$\tan(\lambda_2 - \lambda_1) = \frac{\sin\beta \sin\left(\frac{r}{R_p}\right)\cos\varphi_1}{\cos\left(\frac{r}{R_p}\right) - \sin\varphi_1 \sin\varphi_2}.$$

These formulae can all be derived from the spherical form of the sine and cosine laws using relevant third points.

Notes on longitude, latitude and distance

Given a pair of points by longitude and latitude, converting this into a distance between them can be a surprisingly involved calculation, involving a successively improving model of the shape of the Earth (the geoid).

At the lowest reasonable level of approximation, in which the Earth is considered spherical, points at the same longitude satisfy a formula

$$|\varphi_1 - \varphi_2| = \frac{r}{R_p}$$

where the φs are the latitudes (in radians), r the surface distance between the points and R_p the radius of the earth. As long as r and R_p are in the same units, the choice doesn't matter, but metres are usually to be preferred.

For points at the same latitude, a similar formula applies,

$$|\lambda_1 - \lambda_2| = \frac{r}{R_p \cos \varphi},$$

where the λs are the longitudes and the φ is the common latitude.

In the general case a number of different formulas exist. Among the more popular are the Haversine formula:

$$\frac{r}{R_p} = 2\arcsin\sqrt{\sin^2\frac{|\varphi_1 - \varphi_2|}{2} + \cos\varphi_1\cos\varphi_2\sin^2\frac{|\lambda_1 - \lambda_2|}{2}},$$

the spherical Vincenty formula:

$$\frac{r}{R_p} = \arctan\frac{\sqrt{(\cos\varphi_2\sin|\lambda_1 - \lambda_2|)^2 + (\cos\varphi_1\sin\varphi_2 - \sin\varphi_1\cos\varphi_2\cos|\lambda_1 - \lambda_}}{\sin\varphi_1\sin\varphi_2 + \cos\varphi_1\cos\varphi_2\cos|\lambda_1 - \lambda_2|}$$

and the law of spherical cosines:

$$\frac{r}{R_p} = \arccos(\sin\varphi_1\sin\varphi_2 + \cos\varphi_1\cos\varphi_2\cos|\lambda_1 - \lambda_2|).$$

At short distances linearizations such as Pythagoras can also be used.

Which formulae to choose is a balance between the cost of calculation and the accuracy of the result, which also depends on the specifics of the implementation. For example the two argument (also called `arctan2`) inverse tangent function should be preferred when needed (and available). In general the cheaper formulas have fewer trignometric function evaluations and square root calculations.

For this project, you should assume a spherical Earth and use one of the above approximations, but you may be interested to know that at the next level of approximation, the Earth is considered as an oblate spheriod (i.e. flattened sphere) and the full, iterative version of Vincenty's formulae can be used. Further improvement includes local effects and acknowledges the implications of land elevation, but that sits well outside the scope of this exercise.

Extended functionality

Additional credit will be given if your damage mapper function demonstrates the following extended capabilities:

- The ability to present the software output on a map. The graphics should be designed to be appropriate for use in emergency response and evacuation planning.
- The ability to perform a simple uncertainty analysis that takes as input a small uncertainty on each input parameter and calculates a risk for each affected UK postcode (sector).

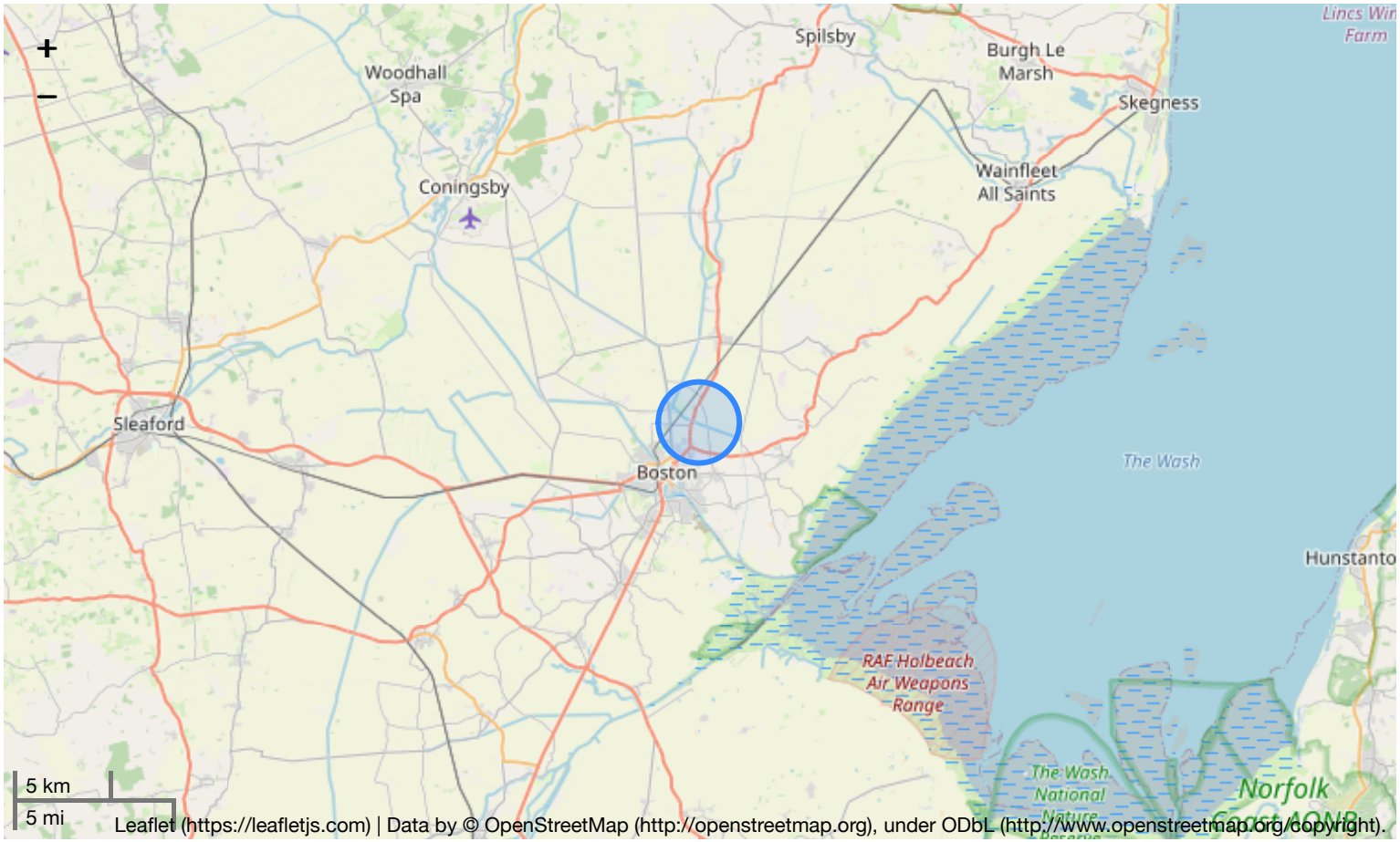
Plotting on a map

As one possible approach, we have provided a function to plot a circle on a map using the `folium` package. You can use `folium` and expand on this function or you may prefer to use a different package. Please check with us that the mapping package you wish to use is permissible before you start.

In [4]:

```
import folium
import armageddon
armageddon.plot_circle(53., 0, 2000.) #Plots a circle of radius 2000 m at the lat, lon: 53., 0.
```

Out[4]:



A NOTE ON PERMITTED PACKAGES

- See `requirements.txt` for preapproved list. Other packages used in lectures may be allowed, but please ask first
- If in doubt, please ask!

```
numpy >= 1.13.0          pytest-timeout
scipy                    sphinx
sympy                    dask
pandas                   distributed
matplotlib               googledrivedownloader
mpltools                 folium
pytest
```

Questions?