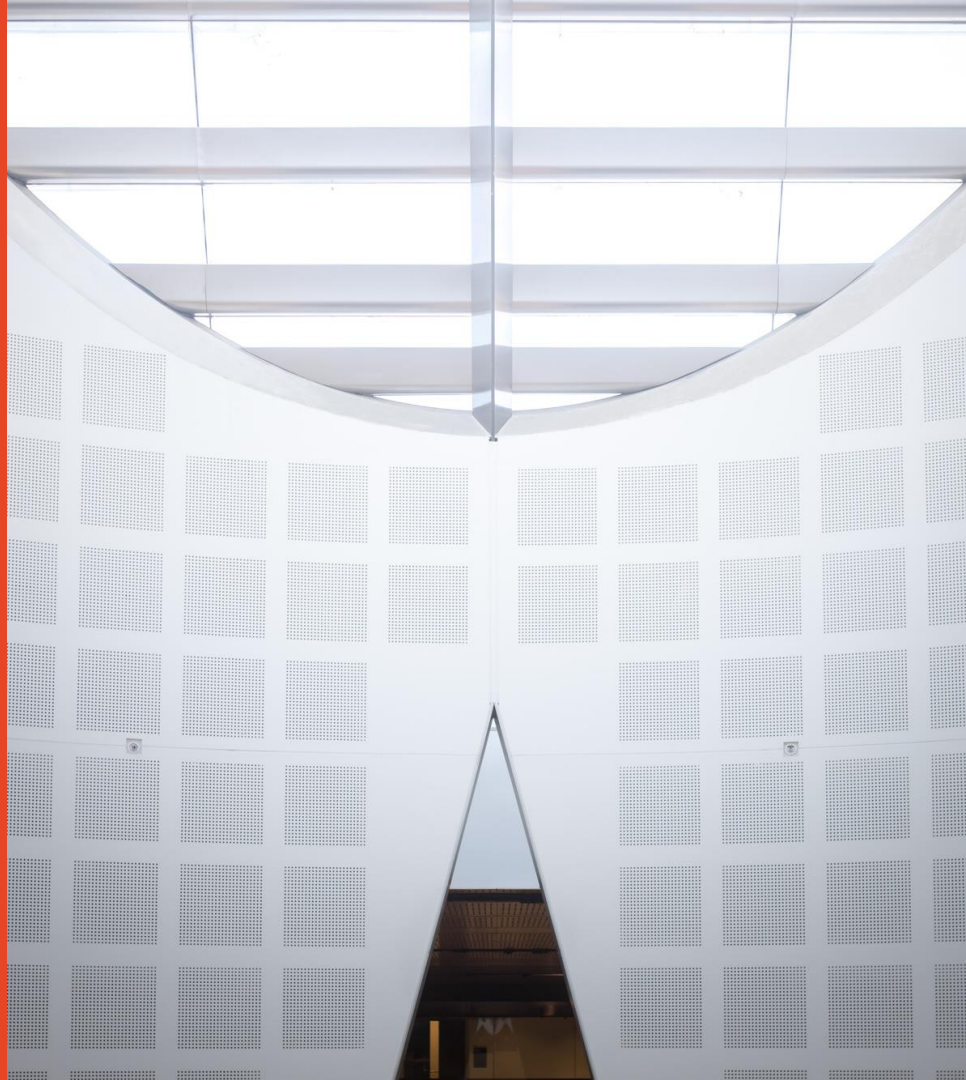
INFO3406
Introduction to Data Analytics
W3: Data Exploration with Python

Presented by
Dr Ali Anaissi
School of IT

THE UNIVERSITY OF SYDNEY



Assignment 1: Project Stage 1

Project stage 1: Data Exploration and Cleaning

Objective

Explore a data set and define a research question based on research/business requirement.

Activities

- Choose a data set
- Explore and summarise data set
- Clean and prepare data
- Define problem

Output

- **2-page report** summarising data, problem and explorative analysis
  - how did you acquire the data?
  - which tools did you use to clean and explore the data set?
  - with title page & references: max 4p

Marking

- 13% of overall mark

Marking Criteria

- Problem Definition
- Acquisition of a relevant Data Set
- Data Preparation (Cleaning & Transformation)
- Data Summarization (Explorative Analysis)
- Report Structure and Style
- Appropriateness of Tools

Detailed marking rubric to be published on Canvas

Suggested timeline for Assignment 1 (Project Stage 1)

- W1: Identify possible data sets
- W2: Identify and Explore possible data sets
- ***W3: Select project data set, define problem, complete exploration***
- W4: Draft summary (problem & exploratory analysis)
- W5: Clean and prepare data
- W6: Descriptive Stats, justification of suitability

Types of projects to consider

- Discover clusters in data
- Learn association rules
- Train a classifier and evaluate prediction accuracy
- Train a regression model and evaluate prediction accuracy

Overview of Week 3

Today: Data Exploration with Python

Objective

Learn Python tools for exploring a new data set programmatically.

Lecture

- Data types, cleaning, preprocessing
- Descriptive statistics, e.g., median, quartiles, IQR, outliers
- Descriptive visualisation, e.g., boxplots, confidence intervals

Readings

- Data Science from Scratch: Ch 4-5

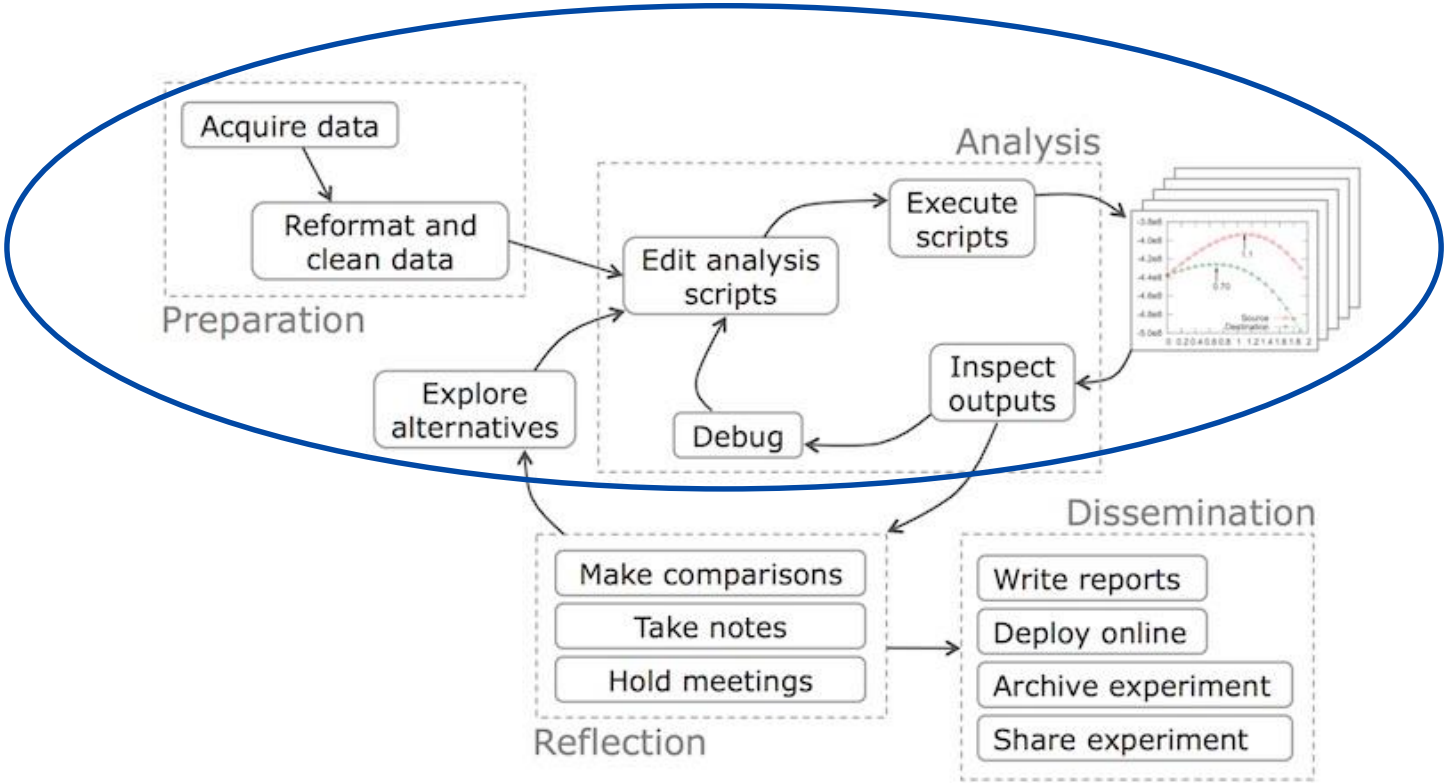
Exercises

- matplotlib: Visualisation
- numpy/scipy: Descriptive stats

TODO in W2

- Grok Python modules 1-4
- Grok SQL modules 1-4
- Explore and select project data

Exploratory analysis workflow



We'll revisit some descriptive questions with Python

- What industries do we know? What would we like to go into?
- What areas of data analytics are considered important?
- How do professional/programming experience compare?
- How does programming experience differ across industries?
- What skills do we know? What would we like to learn?

And look at a text question

- Which industries are most desirable? Do past/future differ?
- What areas are considered most important? Reliable?
- What skills co-occur most? How strong is the association?
- Are there natural clusters corresponding to profiles?
- Is there a significant dependence between experience?
- What terms/topics characterise our DA definitions?

Python and Jupyter Notebooks

Python is great for prototyping

- **Interpreted**: direct execution without compilation
- **Dynamically-typed**: don't have to declare a static type
- **Readable**: easy-to-understand syntax
- **Deployable**: easy to incorporate in applications
- **Productivity**: facilitates rapid, interactive prototyping

Python Recap

- general program syntax
- variables and types
  - integer and float numbers, string types, type conversion
  - list of values (list, array)
- condition statements (if/elif/else)
- for loops, ranges
- functions
  - input(), print(), len(), lower(), upper(), …
  - nesting of functions; example: print(len(*str*.upper()))

Python Import System

- Grok lessons, so far, concentrated on *built-in* functions
- additional functionality available via **import** statement
  - gives access to classes and functions from various 3rd party modules
  - Example: **csv:** comma-separated file format support

```
import csv
for row in csv.reader( ['one,apple,green', 'two,tomato,red'] ):
    print(row[1])
```

  - alternative usages to introduce shortcuts or import only certain functions:

```
import csv as X
from csv import DictReader
```

Python has excellent open-source data libraries

- **scipy**: libraries for scientific and technical computing
- **numpy**: support for large multidimensional arrays and matrices
- **matplotlib**: port of matlab plotting functionality
- **scikit-learn**: machine learning library
- **nltk**: natural language toolkit

- **pandas**: R-like data frame and associated manipulations

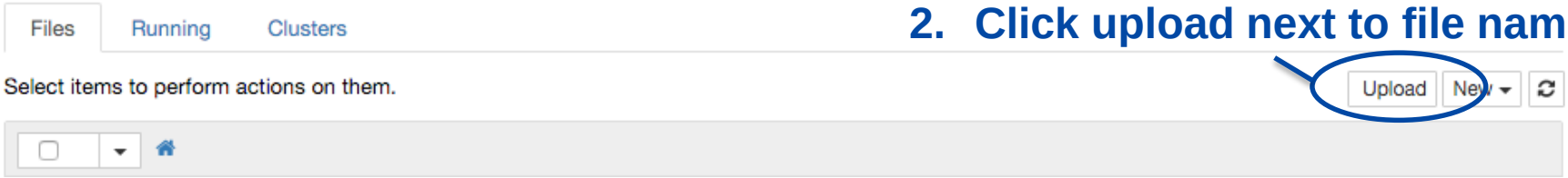
Jupyter notebooks support interactive data science

- IPython interactive command shell offers:
  - Introspection
  - Tab completion
  - Command history
- Jupyter runs in a browser and supports:
  - Sharing and documenting of live code
  - Data cleaning, visualisation, machine learning, …
  - Jupyter's gallery of interesting notebooks: https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks
- SIT provides a Jupyter server which runs Python 3 (v3.6.2)

In the lab: Upload survey data and notebook to Jupyter

1. Download data and notebook from Canvas
2. Login to one of these Jupyter Servers
   - https://soit-ucpu-pro-**X**.ucc.usyd.edu.au (**X** = *1 or 3*)
   - Log in with unikey as username and password
   - Both servers are linked to your ICT shared folder here at Usyd
     - *Note: folders are created at 1st login to one of the SIT lab machines*
3. Upload data and notebook to Jupyter Hub

1. **Click here for file open dialogue**
2. **Click upload next to file name**



Jupyter notebook cells

Markdown cell for formatted text

Data exploration with Python

EXERCISE: Reading and accessing data

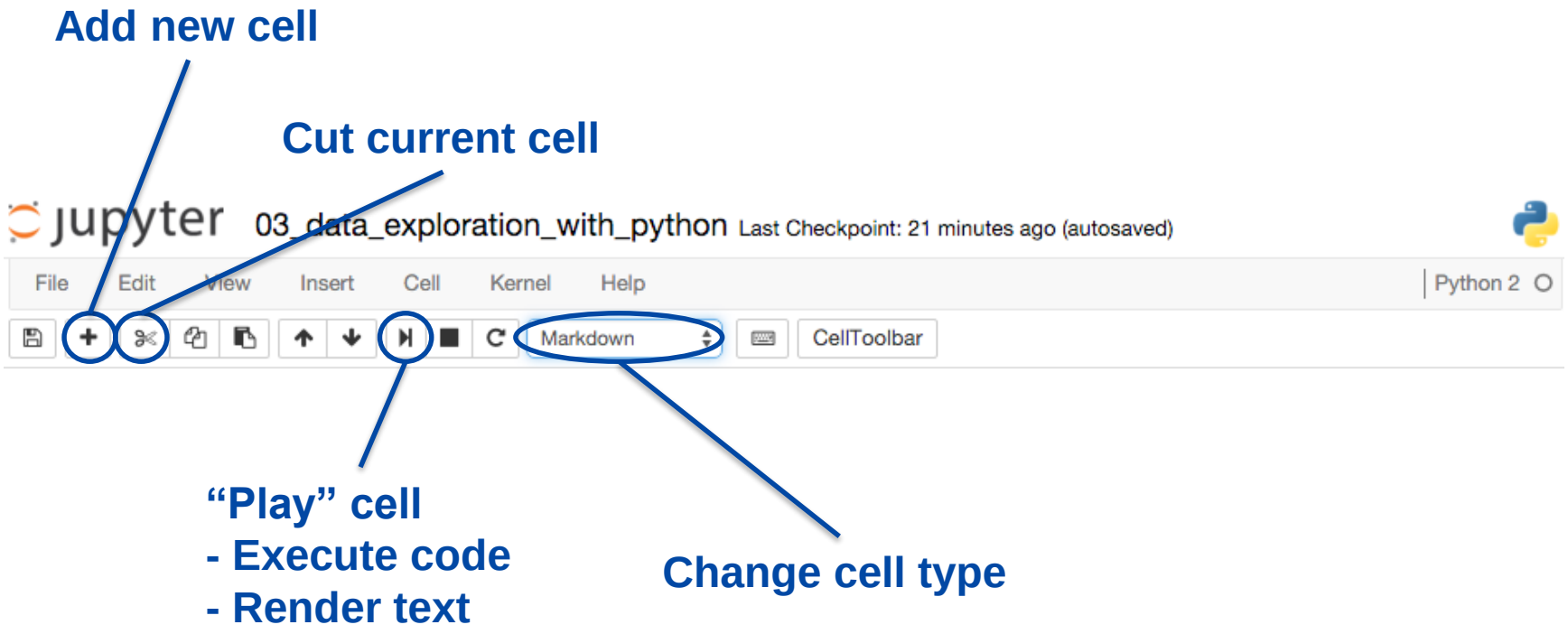
Read the survey response data

The `csv` module supports reading and writing of files in comma-separated values (CSV) and similar formats. We use `DictReader` since the first row of our survey responses file is a header. This produces a list of dictionaries, one dictionary per 57 responses. The `pprint` command below prints the dictionary corresponding the the first response.

In [1]:
```
import csv
import pprint
data = list(csv.DictReader(open('02_ds_survey_responses_raw_20160301.csv')))
pprint.pprint(data[0])
```

Code cell for writing Python commands

Jupyter menu bar



Read data using csv

- Python **csv** module
  - Reads/writes comma-separated values with escaping
  - csv.reader reads rows into arrays
  - csv.DictReader reads rows into ***dictionaries***
- Python **pprint** module
  - pretty print of complex data structures
  - pprint formats a dictionary read by CSV so it's easier to read

```
import csv
import pprint
data = list(csv.DictReader(open('Survey_INFO3406_2018s2 - Form Responses 1.csv')))
pprint.pprint(data[0])
```

Descriptive Statistics

Frequency distributions using collections.Counter

- The **collections** module provides several useful data structures
- Counter
  - Takes a list or other iterable as input and counts its entries
  - Returns a count of how often each item appears

```
from collections import Counter

counts = Counter()
for row in data:
    counts[row[IMPORT_COMMUNICATION]] += 1

print("Distribution of communication importance ratings:")
for k, v in sorted(counts.items()):
    print('{}: {}'.format(k, v))
```

Loop through all data and count communication ratings

Sort and print each k (value) v (count) pair

Calculating the mode with collections.Counter

- Recall that the **mode** is the most frequent value
- We can use the most_common() method to calculate it

Define a mode function

First argument is the data set

Second argument is the column key

```
def mode(data, column_key):
    mode_counter = Counter()
    for row in data:
        mode_counter[row[column_key]] += 1
    return mode_counter.most_common(1)[0][0]

# example on how to use the 'mode' function
print("Communication mode:", mode(data, IMPORT_COMMUNICATION))
```

Calculate the n=1 most common Values, access the first value, Return the value (not the count)

Cleaning data: convert to correct types

- The Python csv module reads everything as string types
- Need to convert as appropriate (e.g., int, float, timestamp)
  - int() creates integer objects, e.g., -1, 101
  - float() creates floating point object, e.g., 3.14, 2.71
  - datetime.strptime() creates datetime objects from strings

A function to convert values in a given column

Use "not a number" (*nan*) as default value
numpy knows to ignore for some stats

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

A function to convert values in a given column

Define *clean* function that cleans given data

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

A function to convert values in a given column

Get original value from row
Set new value to default

Attempt conversion
catching errors

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

A function to convert/clean values in a given column

list of known strings
and their numerical equivalent

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5}
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

replace known strings
with valid number

Cleaning float data

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5}
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

Call for professional and programming experience columns

Cleaning date data

Format string for US timestamp from Google Sheets, e.g., "3/1/2016 15:59:56"

```
from datetime import datetime
FMT = "%m/%d/%Y %H:%M:%S"
def str_to_time(s):
    if isinstance(s, datetime):
        return s
    return datetime.strptime(s, FMT)
clean(data, TIMESTAMP, str_to_time, DEFAULT_VALUE)
```

Return a datetime object for string s

Call for timestamp column

Central tendancy and dispersion with numpy

- **Numpy** provides various statistics for numeric data
- Median, percentiles, mean, standard deviation, etc
- nan* versions calculate same statistics, ignoring NaN values

- Reference page for numpy statistics:
  http://docs.scipy.org/doc/numpy/reference/routines.statistics.html

Calculating central tendency and dispersion

Calculate stats for professional and programming experience

```
import numpy as np
for column_key in [BACKGROUND_YEARS_PROFESSIONAL, BACKGROUND_YEARS_PROGRAMMING]:
    v = [row[column_key] for row in data] # grab values
    print(column_key.upper())
    print("* Min..Max: {}..{}".format(np.nanmin(v), np.nanmax(v)))
    print("* Range: {}".format(np.nanmax(v)-np.nanmin(v)))
    print("* Mean: {}".format(np.nanmean(v)))
    print("* Standard deviation: {}".format(np.nanstd(v)))
    print("* Median: {}".format(np.nanmedian(v)))
    q1 = np.nanpercentile(v, 25)
    print("* 25th percentile (Q1): {}".format(q1))
    q3 = np.nanpercentile(v, 75)
    print("* 75th percentile (Q3): {}".format(q3))
    iqr = q3-q1
    print("* IQR: {}".format(iqr))
```

```
v = [ ]
for row in data:
    v.append(row[column_key])
```

Calculate min/max, range, mean, standard deviation, median, 25/75th percentiles, inter-quartile range

Binning and Histograms

Create histogram of 7 bins ranging from 0 to 35

```
v = []
for row in data:
    v.append(row[BACKGROUND_YEARS_PROFESSIONAL])

freqs, bins = np.histogram(v, bins=7, range=(0,35)) # calculate frequencies and bin start/end
for i, freq in enumerate(freqs):
    # Note that bins[i] <= bin_values < bins[i+1]
    bin_str = '[{}..{}]'.format(int(bins[i]), int(bins[i+1]))
    print(bin_str, ':', freq)
```

Visualisation

Visualising data with matplotlib

- Matplotlib provides functionality for creating various plots
- Bar charts, line charts, scatter plots, etc

- Reference page for pyplot:
  http://matplotlib.org/api/pyplot_api.html
- Documentation:
  http://matplotlib.org/contents.html

Creating a bar chart

Use Counter to get frequency distribution
Reorder according to IMPORT_KEYS
Add default count of 0 for missing values

```
from collections import OrderedDict
IMPORT_KEYS = ['1', '2', '3', '4', '5']
def make_importance_plot(data, column_key, title):
    c = Counter(row[column_key] for row in data)
    d = OrderedDict([(k,c[k]) if k in c else (k,0) for k in IMPORT_KEYS])
    # bars are by default width 0.8, so we'll add 0.1 to the left coordinates
    xs = [i+0.1 for i,_ in enumerate(IMPORT_KEYS)]
    plt.bar(xs, d.values())
    plt.ylabel('Number of responses')
    plt.axis([0,5,0,35])
    plt.title(title)
    plt.xticks([i + 0.5 for i, _ in enumerate(IMPORT_KEYS)], IMPORT_KEYS)
    plt.show()
for a in IMPORT_AREAS:
    title = 'Importance of {}'.format(a.lower())
    make_importance_plot(data, a, title)
```

Creating a bar chart

List of indices for x axis

```
from collections import OrderedDict
IMPORT_KEYS = ['1', '2', '3', '4', '5']
def make_importance_plot(data, column_key, title):
    c = Counter(row[column_key] for row in data)
    d = OrderedDict([(k,c[k]) if k in c else (k,0) for k in IMPORT_KEYS])
    # bars are by default width 0.8, so we'll add 0.1 to the left coordinates
    xs = [i+0.1 for i, _ in enumerate(IMPORT_KEYS)]
    plt.bar(xs, d.values())
    plt.ylabel('Number of responses')
    plt.axis([0,5,0,35])
    plt.title(title)
    plt.xticks([i + 0.5 for i, _ in enumerate(IMPORT_KEYS)], IMPORT_KEYS)
    plt.show()
for a in IMPORT_AREAS:
    title = 'Importance of {}'.format(a.lower())
    make_importance_plot(data, a, title)
```

Create bar chart

Set axis ranges:
x: 0..5; y: 0..35

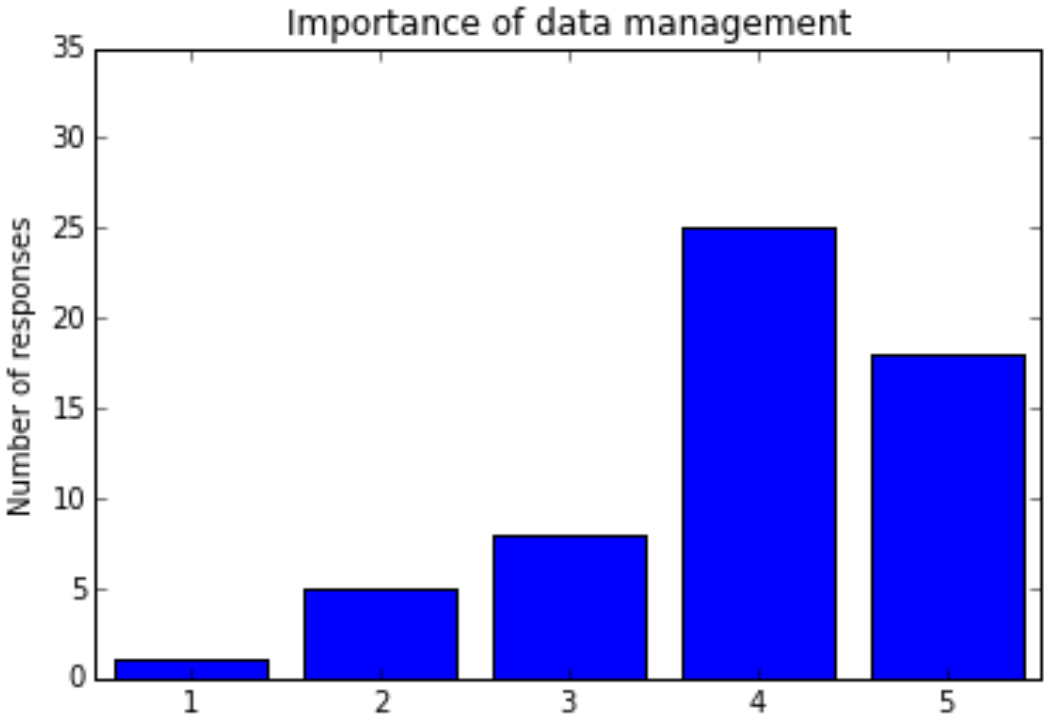
Creating a bar chart

```
from collections import OrderedDict
IMPORT_KEYS = ['1', '2', '3', '4', '5']
def make_importance_plot(data, column_key, title):
    c = Counter(row[column_key] for row in data)
    d = OrderedDict([(k,c[k]) if k in c else (k,0) for k in IMPORT_KEYS])
    # bars are by default width 0.8, so we'll add 0.1 to the left coordinates
    xs = [i+0.1 for i,_ in enumerate(IMPORT_KEYS)]
    plt.bar(xs, d.values())
    plt.ylabel('Number of responses')
    plt.axis([0,5,0,35])
    plt.title(title)
    plt.xticks([i + 0.5 for i, _ in enumerate(IMPORT_KEYS)], IMPORT_KEYS)
    plt.show()
for a in IMPORT_AREAS:
    title = 'Importance of {}'.format(a.lower())
    make_importance_plot(data, a, title)
```

Center x ticks under bars, and pass in labels

Make a plot for each area

A bar chart for data management ratings



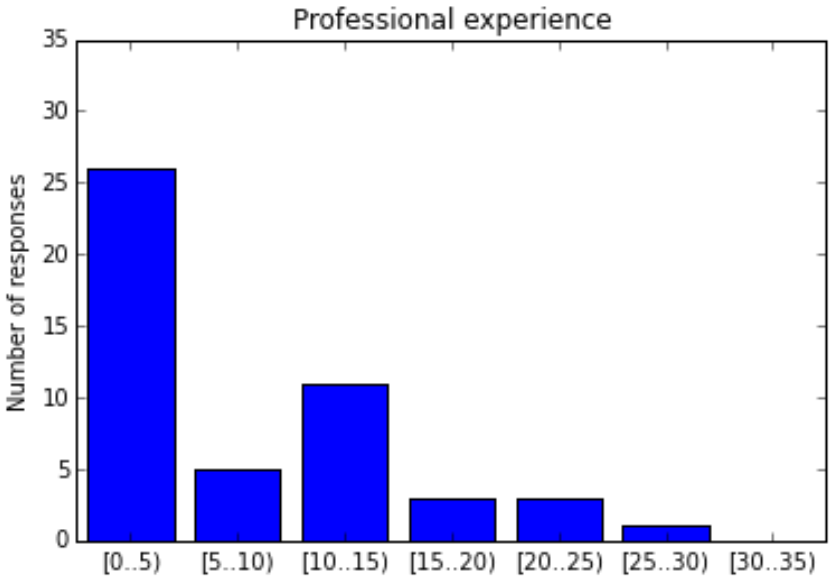
Plotting a histogram

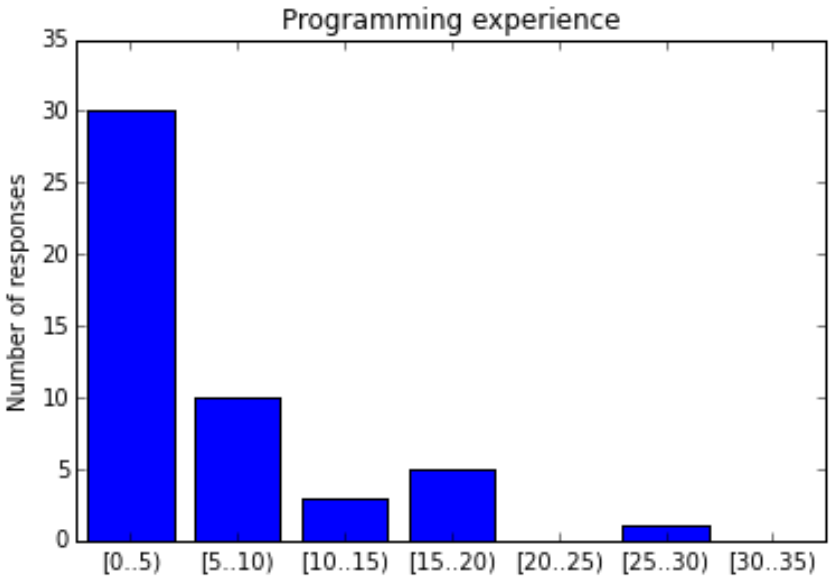
Yield bin_label, frequency pairs

```
def iter_histogram(data, column_key):
    v = [row[column_key] for row in data] # grab values
    freqs, bins = np.histogram(v, bins=7, range=(0,35))
    for i, freq in enumerate(freqs):
        yield ('[{}..{})'.format(int(bins[i]), int(bins[i+1])), freq)

def make_histogram_plot(data, column_key, title):
    d = OrderedDict(iter_histogram(data, column_key))
    keys = list(d.keys())
    xs = [i+0.1 for i,_ in enumerate(keys)]
    plt.bar(xs, d.values())
    plt.ylabel('Number of responses')
    plt.axis([0,7,0,35])
    plt.title(title)
    plt.xticks([i + 0.5 for i, _ in enumerate(keys)], keys)
    plt.show()

make_histogram_plot(data, BACKGROUND_YEARS_PROFESSIONAL, 'Professional experience')
make_histogram_plot(data, BACKGROUND_YEARS_PROGRAMMING, 'Programming experience')
```

Create plots for professional and programming experience

Histograms for experience (bin size 5 years)



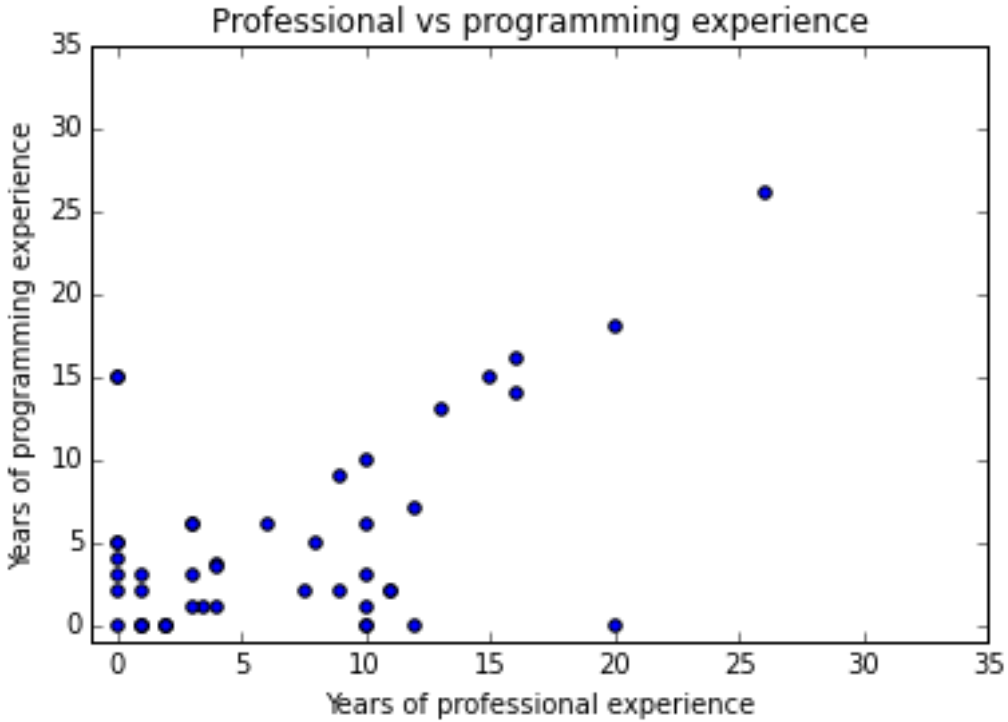


Creating a scatter plot

Create scatter plot

```
professional_experience = [row[BACKGROUND_YEARS_PROFESSIONAL] for row in data]
programming_experience = [row[BACKGROUND_YEARS_PROGRAMMING] for row in data]
plt.scatter(professional_experience, programming_experience)
plt.title('Professional vs programming experience')
plt.xlabel('Years of professional experience')
plt.ylabel('Years of programming experience')
plt.axis([-1,35,-1,35])
plt.show()
```

A scatter plot comparing experience

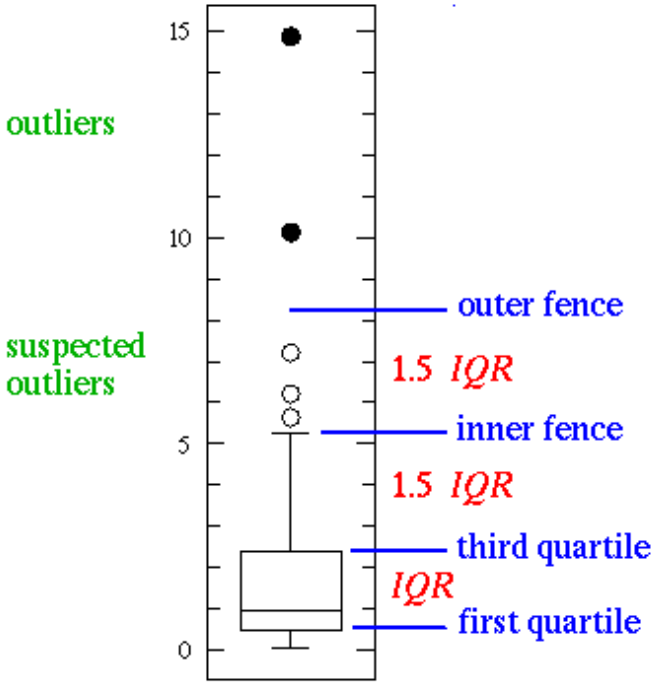


Box plots and correlation

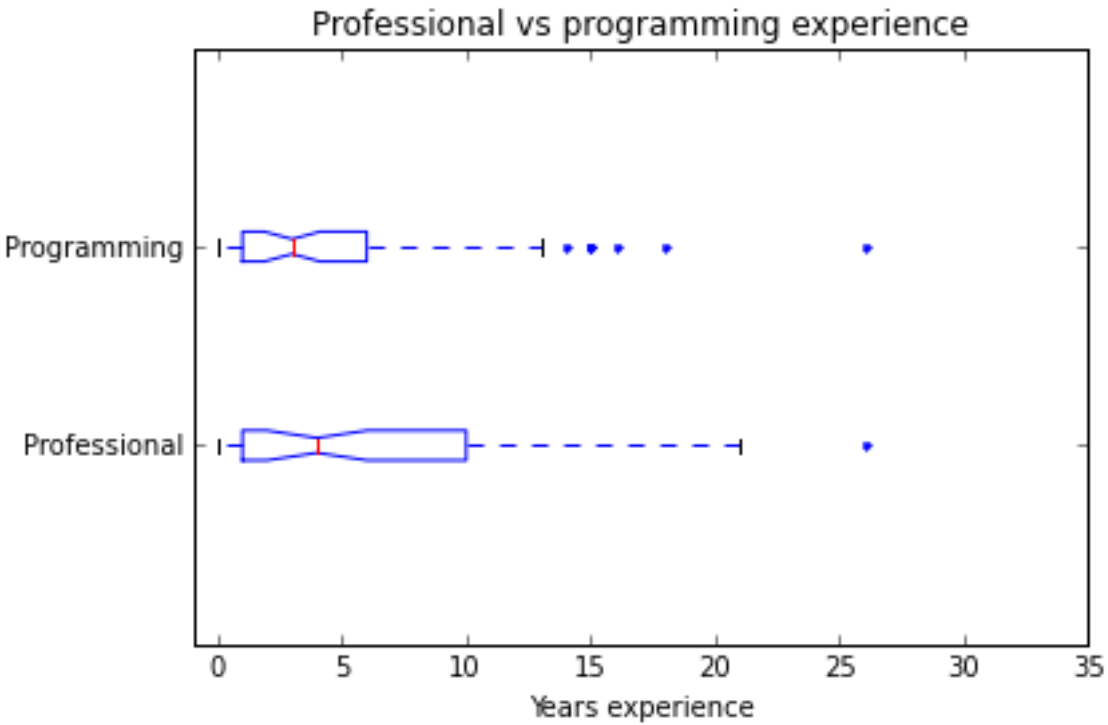
Using boxplots to compare distributions

- Mean and stdev are not informative when data is skewed
- **Box plots** summarise data based on 5 numbers:
  - Lower inner fence – Q1–1.5*IQR
  - First quartile (Q1) – equivalent to 25th percentile
  - Median (Q2) – equivalent to 50th percentile
  - Third quartile (Q3) – equivalent to 75th percentile
  - Upper inner fence – Q3+1.5*IQR
- Values outside fences are outliers
- Sometimes include outer fences at 3*IQR

Box Plots illustrated



A box plot comparing experience distributions



Using correlation statistics to measure dependence

- Scipy includes various correlation statistics
  - Pearson's *r* for two normally distributed variables
  - Spearman's *rho* for ratio data, ordinal data, etc (rank-order correlation)
  - Kendall's *tau* for ordinal variables

- List of various scipy statistics including correlation coefficients: http://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html

Calculating correlation

Since correlation is paired, grab values where both variables are defined

```
from scipy import stats
# only keep rows where both professional and programming experience are defined
prof, prog = [], []
for row in data:
    if row[BACKGROUND_YEARS_PROFESSIONAL] is np.nan:
        continue # ignore rows with no value for professional experience
    elif row[BACKGROUND_YEARS_PROGRAMMING] is np.nan:
        continue # ignore rows with no value for programming experience
    else:
        prof.append(row[BACKGROUND_YEARS_PROFESSIONAL])
        prog.append(row[BACKGROUND_YEARS_PROGRAMMING])
print("Pearson (r, p): {}".format(stats.pearsonr(prof, prog)))
print(stats.spearmanr(prof, prog))
```

Calculate Person's r and Spearman's rho

Text Data

THE UNIVERSITY OF SYDNEY

A simple whitespace tokeniser

```
def tokenise(text):
    for word in text.lower().split():
        yield word.strip('.,')

def is_valid_word(w):
    if w == '':
        return False
    else:
        return True

def iter_ds_def_words(d):
    for row in d:
        for word in tokenise(row[GOALS_DEFINITION]):
            if is_valid_word(word):
                yield word

from collections import Counter
c = Counter(iter_ds_def_words(data))
c
```

Convert text string to lower case and split on whitespace
Remove leading/trailing '.' and ','

Ignore empty strings

Yield each word token from each definition

Count words

Removing stop words

```
STOP_WORDS = frozenset([ # http://www.nltk.org/book/ch02.html#stopwords_index_term
    'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
    'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
    'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
    'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
    'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
    'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
    'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
    'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
    'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
    'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
    'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now'
    ])
def is_valid_word(w):
    if w == '':
        return False
    if w.lower() in STOP_WORDS:
        return False
    else:
        return True

c = Counter(iter_ds_def_words(data))
c
```

Ignore empty strings and words in stop list

Plotting most frequent words

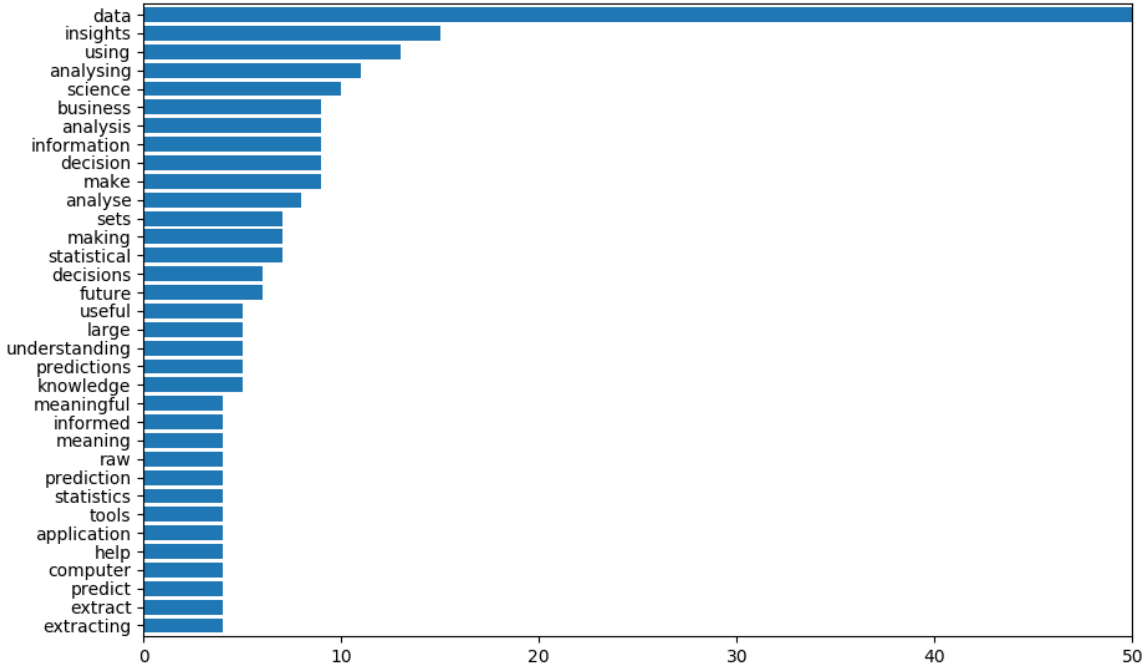
Yield words and their frequencies if they occur 4 or more times

Sort by frequency

Create a horizontal bar chart

```
import operator

def iter_word_freqs(d, min_freq=4):
    c = Counter(iter_ds_def_words(d))
    for term, freq in c.items():
        if freq >= min_freq:
            yield term, freq

d = OrderedDict([(k,v) for k,v in sorted(iter_word_freqs(data), key=operator.itemgetter(1))])
ys = [i+0.5 for i,_ in enumerate(d)]
plt.barh(ys, d.values(), align='center')
plt.yticks(ys, list(d.keys()))
plt.axis([0,50,0-0.1,len(d)+0.1])
plt.show()
```

A term frequency bar chart



Review

Notes

- Python a good example of a scripting language for DS
- programmatic approaches allow for more powerful / flexible data preparation and analysis,
  - and more control on the visualisations
- Many useful support libraries available in the Python ecosystem
  - numpy, scipy, matplotlib

Additional reading (not examinable)

- matplotlib API reference
  http://matplotlib.org/api/pyplot_api.html
- NumPy and SciPy documentation
  http://docs.scipy.org/doc/
- Data Analysis with Python (O'Reilly)
  http://shop.oreilly.com/product/0636920023784.do

Participation

Requirements

- Submit code at end of each week
- **Jupyter Notebooks:**
  - The various exercises have placeholder cells marked as TODO:

```
# TODO: replace the content of this cell
raise NotImplementedError
```

  - The content of these cells needs to be replaced with your own solution => basis for participation marking

Output

- Code/spreadsheets from exercises
- Each week's participation assessed as: all done, partially done, no participation

Marking

- 10% of overall mark

Submit your Jupyter notebooks in Canvas by next Friday

Next Time

Next week: Cleaning and storing data

Objective

How to clean and prepare a data set for effective analysis and for storage in a database.

Lecture

- ETL: extract, transform and load
- CSV and SQL

Readings

- Data Science from Scratch: Ch 10 (start) + Ch 23

Exercises

- Preparing CSV data for storage
- Storing data in a database

TODO in W3

- Grok Python modules
- Grok SQL modules
- Explore and select project data