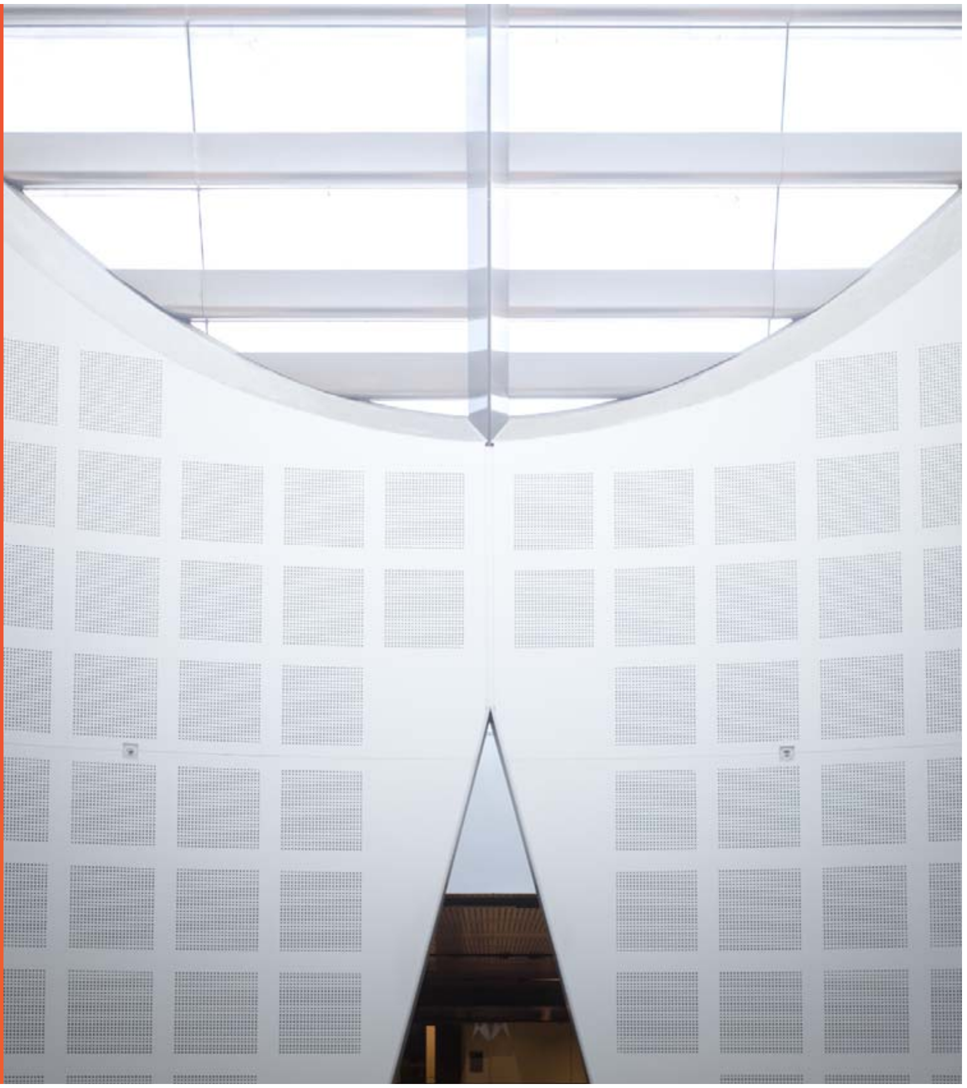
DATA2001: Data Science: Big Data and Data Diversity

Data analysis with Python

Presented by Dr. Matloob Khushi
School of IT

THE UNIVERSITY OF SYDNEY



Python Recap

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

Data analysis 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/W3

– Grok Python modules
– Explore the survey data

Preliminaries: Types of Data

THE UNIVERSITY OF SYDNEY

Data Types

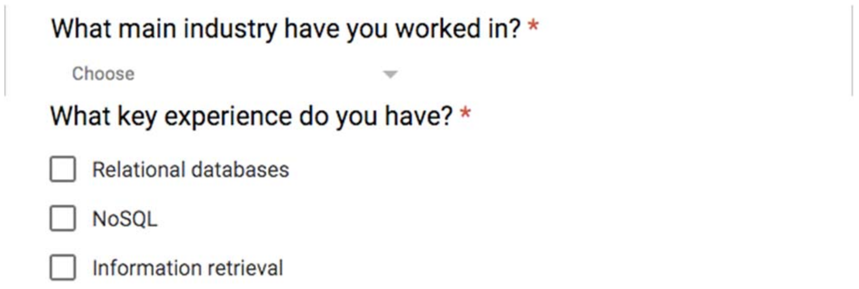
- Text
- Images
- Videos
- Categorical
  - Nominal
    - Dichotomous
  - Ordinal
- Quantitative
  - Interval
  - Ratio

Categorical Data

– A categorical variable is also known as a discrete or qualitative variable and has two or more categories. It is further divided into two variants, nominal and ordinal. These variables are typically coded as numerical values.

Nominal Data

– This is an unordered category data. This type of variable may be coded in numeric form but these numerical values have no mathematical interpretation and are just labeling to denote categories. For example, colours: black, red and white can be coded as 1, 2 and 3.



What main industry have you worked in? *

Choose

What key experience do you have? *

☐ Relational databases

☐ NoSQL

☐ Information retrieval

– Values are names
– No ordering is implied
– Eg jersey numbers

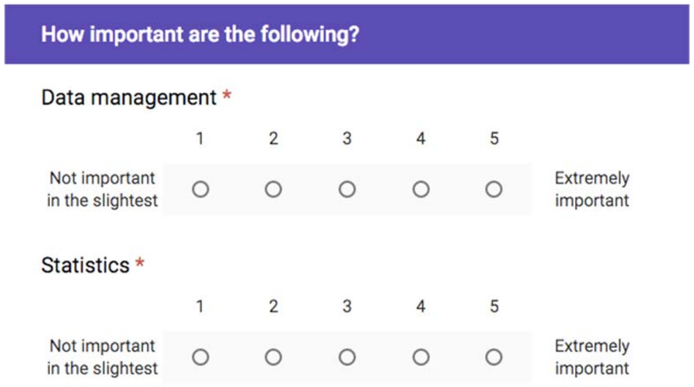
Dichotomous Data

– A dichotomous is a type of nominal data that can only have two possibile values, e.g. true or false, or presence or absence. These are also sometimes referred as binary or Boolean variables.

– True (1) or false (0)
– Gender: male / female

Ordinal Data

- This is ordered categorical data in which there is strict monotonic order. For example, human height (small, medium and high) can be coded into numbers small = 1, medium = 2, high = 3.



How important are the following?

Data management *

	1	2	3	4	5	
Not important in the slightest	○	○	○	○	○	Extremely important

Statistics *

	1	2	3	4	5	
Not important in the slightest	○	○	○	○	○	Extremely important

- Values are ordered
- No distance is implied
- Eg rank, agreement

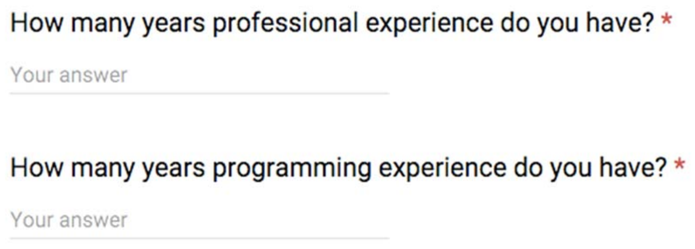
Interval Data

It is a variable in which the interval between values has meaning and there is no true zero value.

- Values encode differences
- equal intervals between values
- No true zero
- Addition is defined
- Eg Celsius temperature

Ratio Data

It is variable that has a true value of zero and represents the total absence of the variable being measured. For example, it makes sense to say a Kelvin temperature of 100 is twice as hot as a Kelvin temperature of 50 because it represents twice as much the thermal energy (unlike Fahrenheit temperatures of 100 and 50).



How many years professional experience do you have? *

Your answer

How many years programming experience do you have? *

Your answer

- Values encode differences
- Zero is defined
- Multiplication defined
- Ratio is meaningful
- Eg length, weight, income

Calculating descriptive statistics

- ***Median*** and ***percentiles*** good here too
- ***Mean*** is the sum of values divided by the number of values:

$$\frac{\sum X_i}{N}$$

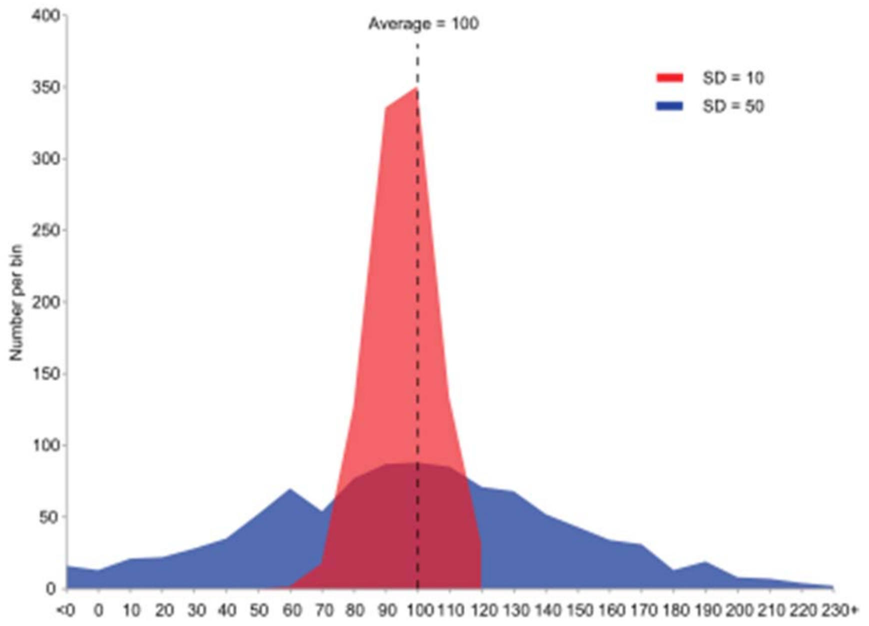
- ***Variance:***

$$\frac{\sum (X_i - mean)^2}{N-1}$$

- ***Standard deviation:***

$$\sqrt{\text{var}\,iance}$$

What does variance and standard deviation tell us?



Samples from two populations with the same mean but different variances. The red population has mean 100 and variance 100 (SD=10) while the blue population has mean 100 and variance 2500 (SD=50).

https://en.wikipedia.org/wiki/Variance

Levels of Measurement

	Nominal	Ordinal	Interval	Ratio
Countable	✓	✓	✓	✓
Order defined		✓	✓	✓
Difference defined (addition, subtraction)			✓	✓
Zero defined (multiplication, division)				✓

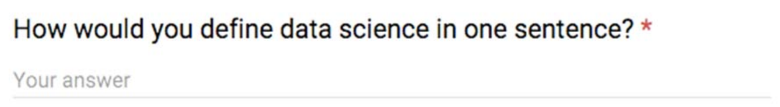
Measures of Central Tendency

	Nominal	Ordinal	Interval	Ratio
Mode	✓	✓	✓	✓
Median		✓	✓	✓
Mean			✓	✓

Measures of Dispersion

	Nominal	Ordinal	Interval	Ratio
Counts / Distribution	✓	✓	✓	✓
Minimum, Maximum		✓	✓	✓
Range		✓	✓	✓
Percentiles		✓	✓	✓
Standard deviation, Variance			✓	✓

What about text data?


How would you define data science in one sentence? *

Your answer

- Not defined as traditional data type in statistics
- Requires interpretation, coding or conversion
- More in future lectures…

Data Acquisition and Cleaning

THE UNIVERSITY OF SYDNEY

Data Acquisition – Where does data come from?

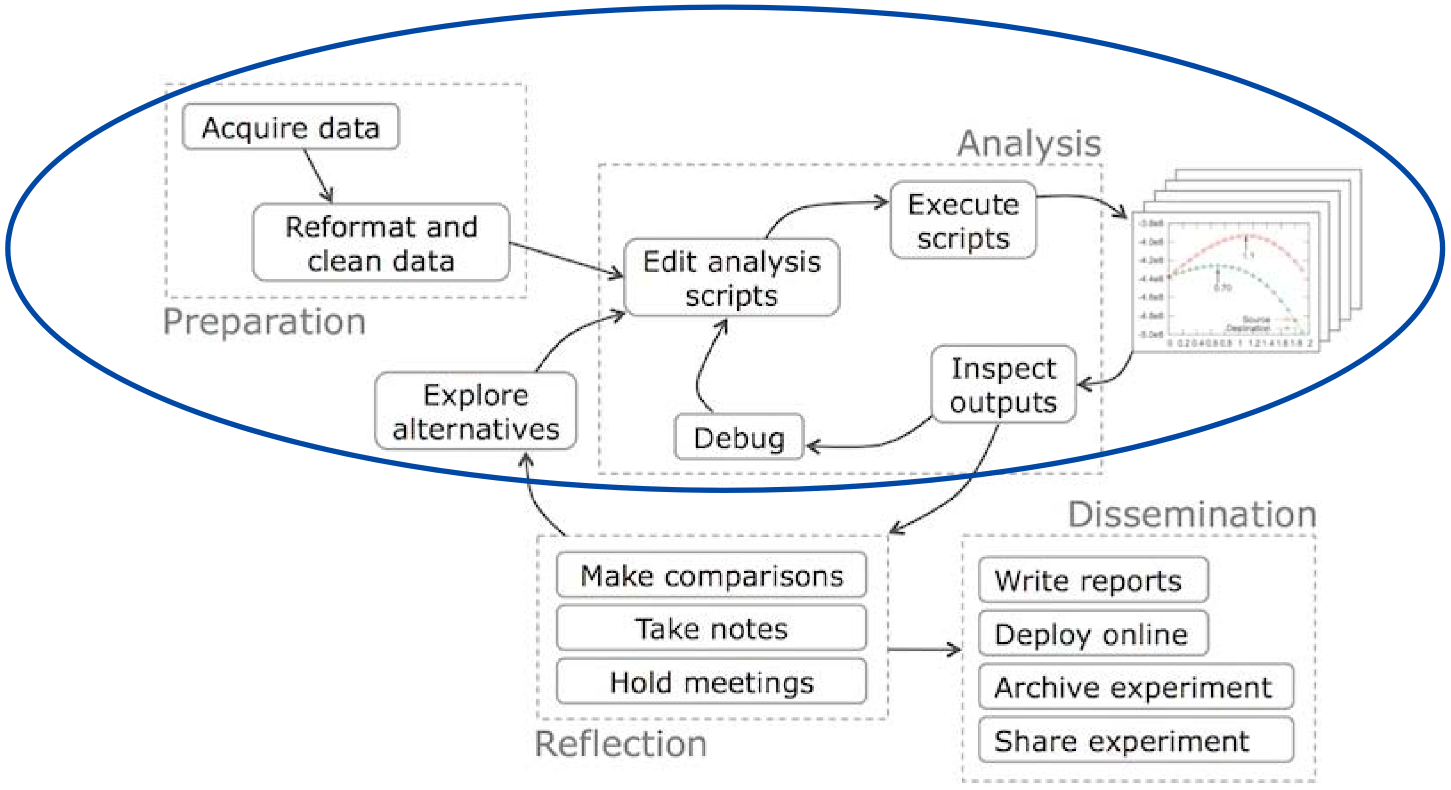
- File Access
  - You or your organisation might already have a data set, or a colleagues provides you access to data.
  - Or: Web Download from an online data server
  - Typical exchange formats: CSV, Excel, sometimes also XML
- Programmatically
  - Scrapping the web (HTML)
  - or using APIs of Web Services (XML/JSON)
     -> Cf. textbook, Ch 9
- Database Access
- Collect data yourself, eg. via a survey

This week: Using data from our online survey

Cleaning and Transforming Data

- Real data is often '*dirty*'
- Important to do some data cleaning and transforming first
- Typical steps involved:
  - type and name conversion
  - filtering of missing or inconsistent data
  - unifying semantic data representations
  - matching of entries from different sources
- Later also:
  - **Rescaling** and optional dimensionality reduction

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 science 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 DS definitions?

Exercise: Upload survey data and notebook to Jupyter

1. Download data and notebook from Canvas page
2. Login to one of these Jupyter Servers
   - https://soit-ucpu-pro-1.ucc.usyd.edu.au
   - https://soit-ucpu-pro-2.ucc.usyd.edu.au
   - https://soit-ucpu-pro-3.ucc.usyd.edu.au
   - https://ucpu1.ug.it.usyd.edu.au
   - 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

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

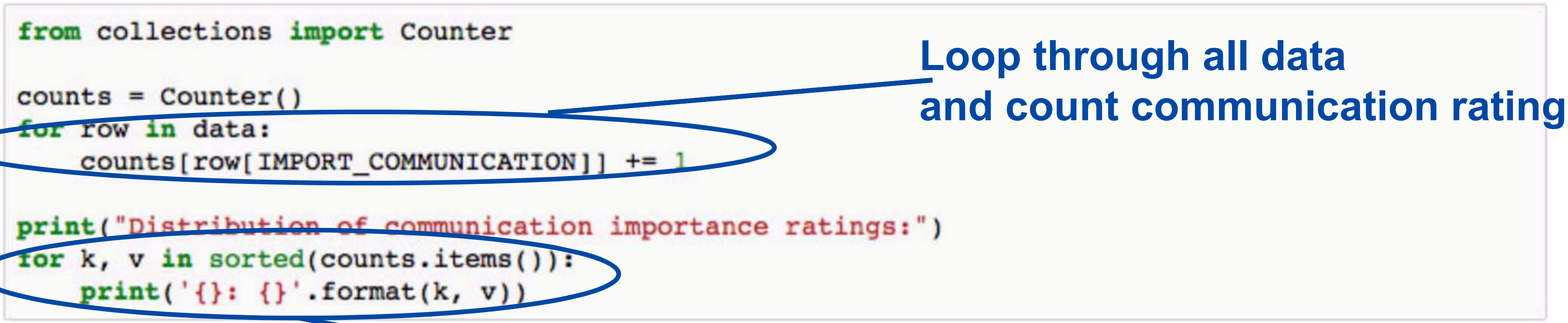
```
import csv
import pprint
data = list(csv.DictReader(open('ds_survey_responses.csv')))
pprint.pprint(data[0])
```

Descriptive Statistics

THE UNIVERSITY OF SYDNEY

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

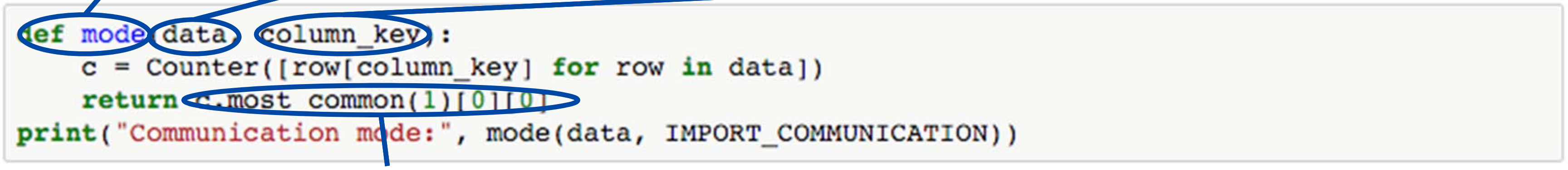
Calculating the mode with collections.Counter

– 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):
    c = Counter([row[column_key] for row in data])
    return c.most_common(1)[0][0]
print("Communication mode:", mode(data, IMPORT_COMMUNICATION))
```

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

Frequency distribution and mode

- Calculate frequency distribution using Counter
  - calculate distributions of industries
- Calculate mode using Counter
  - calculate the mode of industries

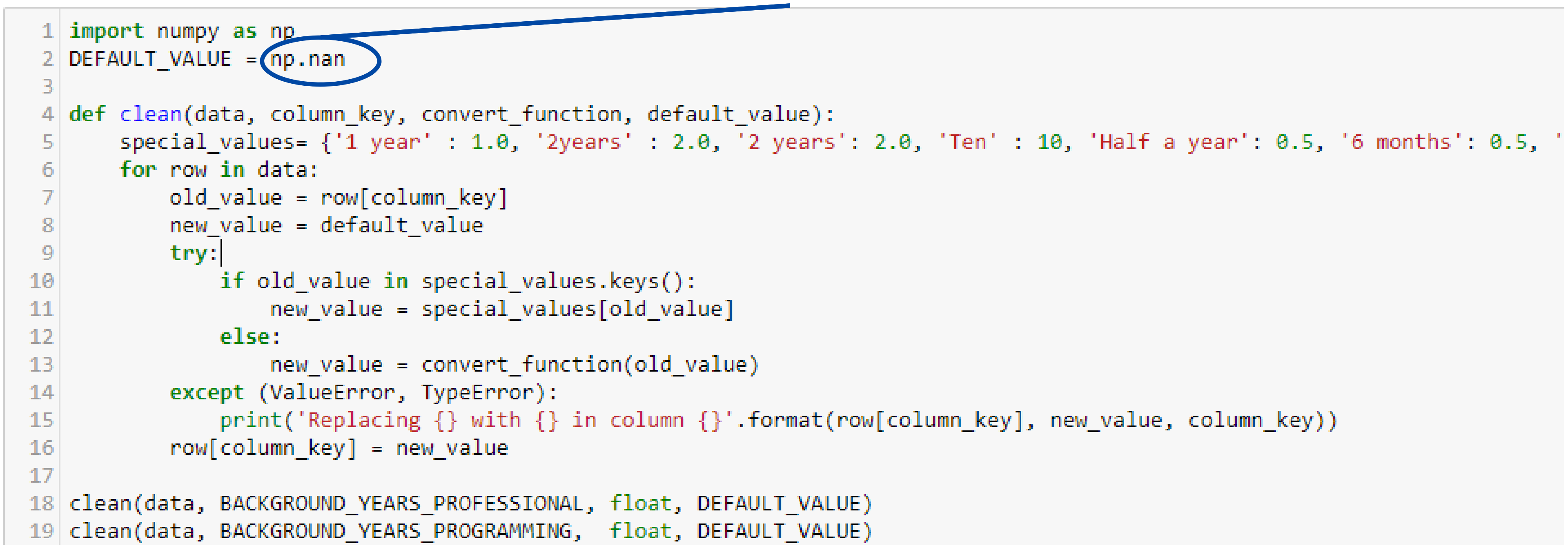
Example in Jupyter

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

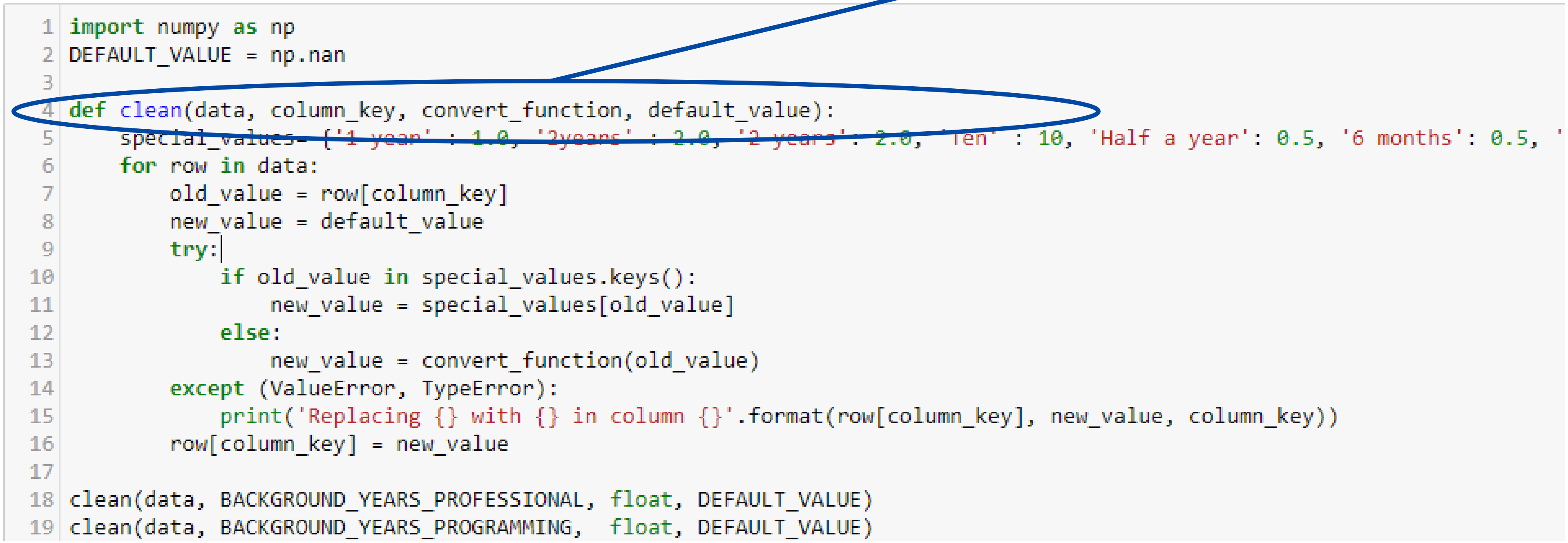
Use “not a number” (*nan*) as default value numpy knows to ignore for some stats



```python
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)
```

A function to convert values in a given column

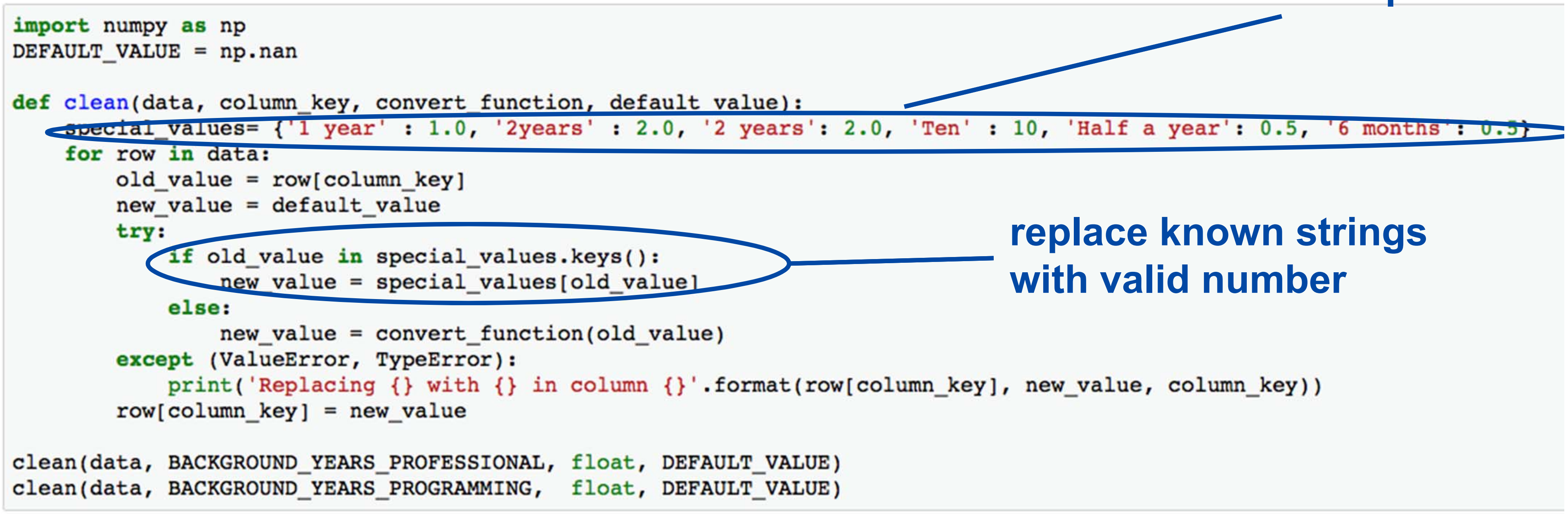
Define *clean* function that cleans given data



```python
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)
```

A function to convert/clean values in a given column

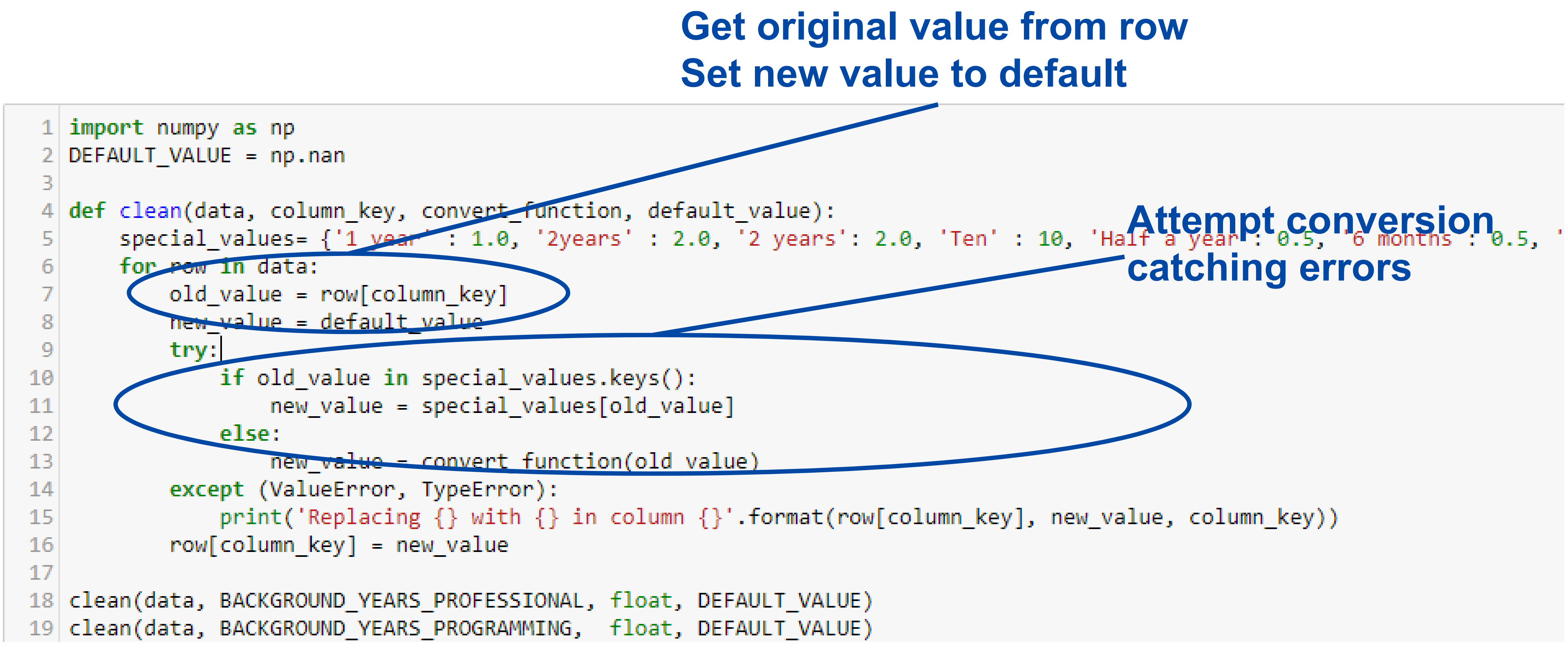
list of known strings
and their numerical equivalent



```python
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)
```

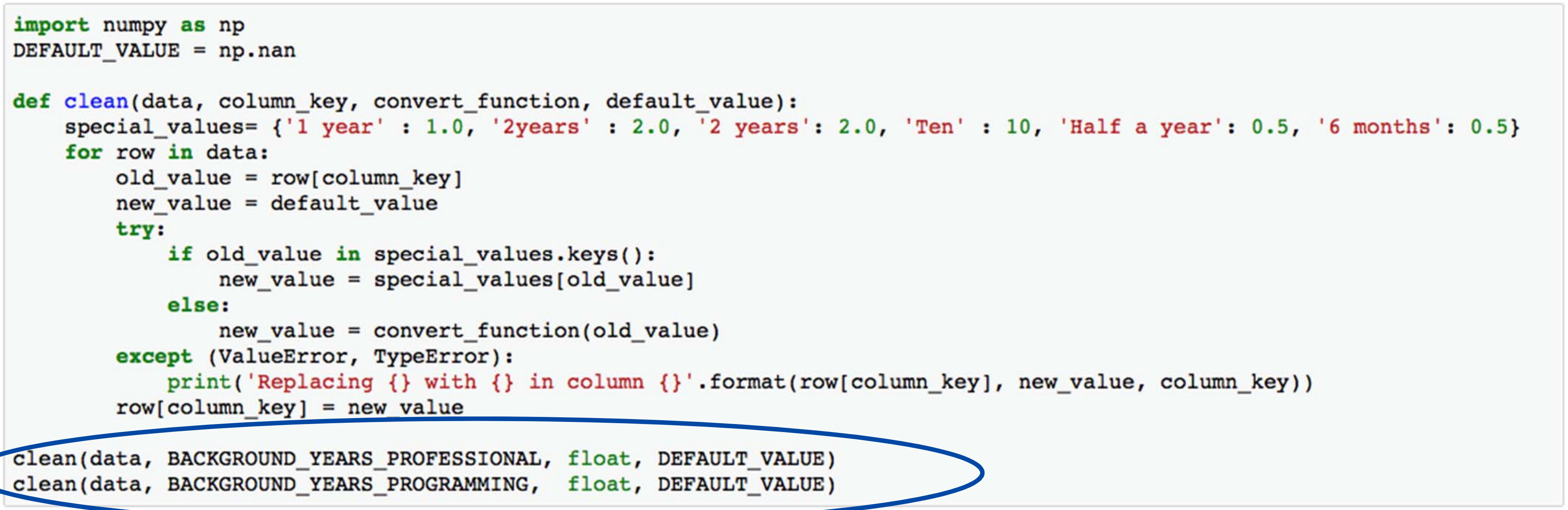
replace known strings
with valid number

A function to convert values in a given column


Get original value from row
Set new value to default
Attempt conversion catching errors

```python
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)
```

Cleaning float data



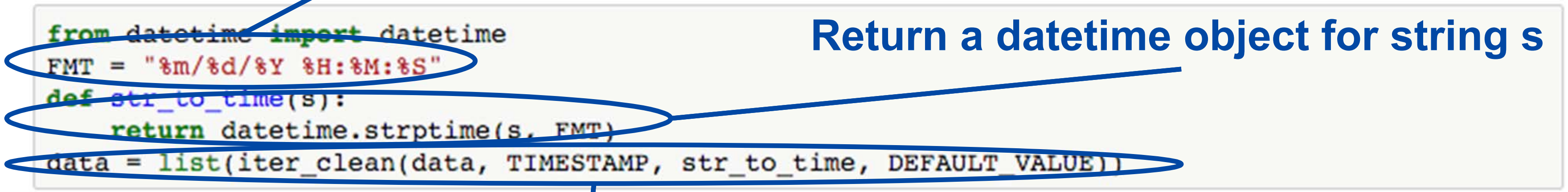
```python
import numpy as np
DEFAULT_VALUE = np.nan

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 float data

Format string for US timestamp from e.g.,“3/1/2016 15:59:56”

Return a datetime object for string s



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

Call for timestamp column

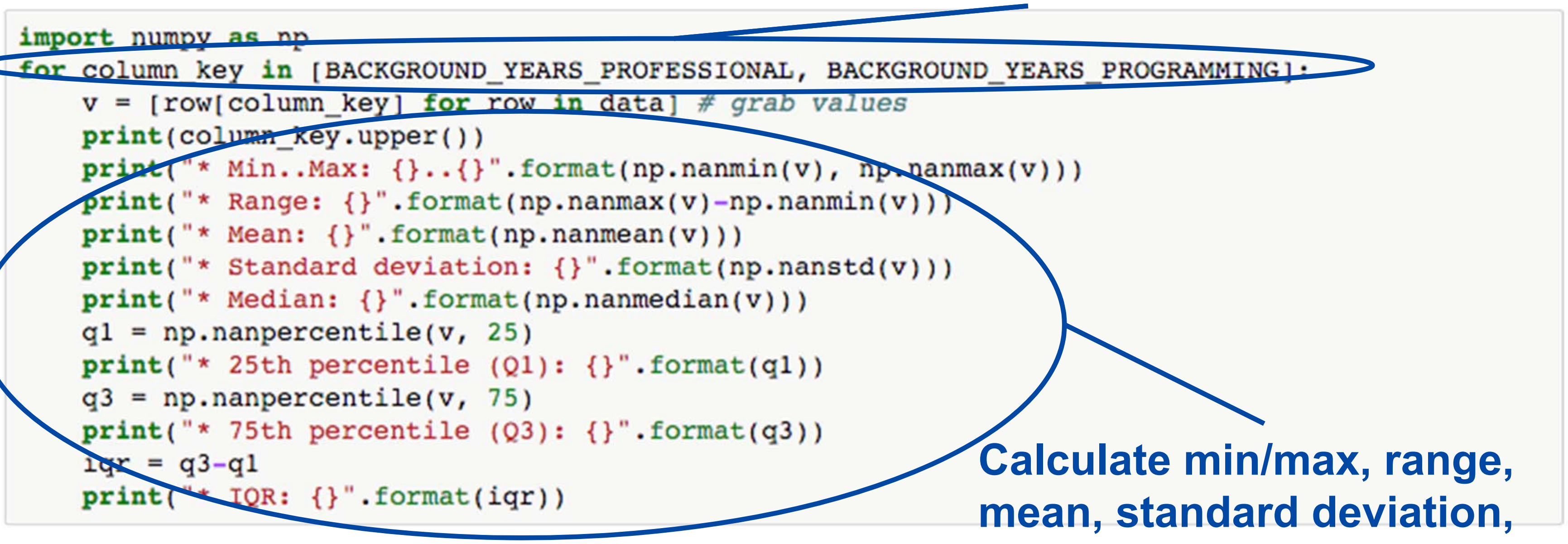
Central tendancy and dispersion with numpy

- **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

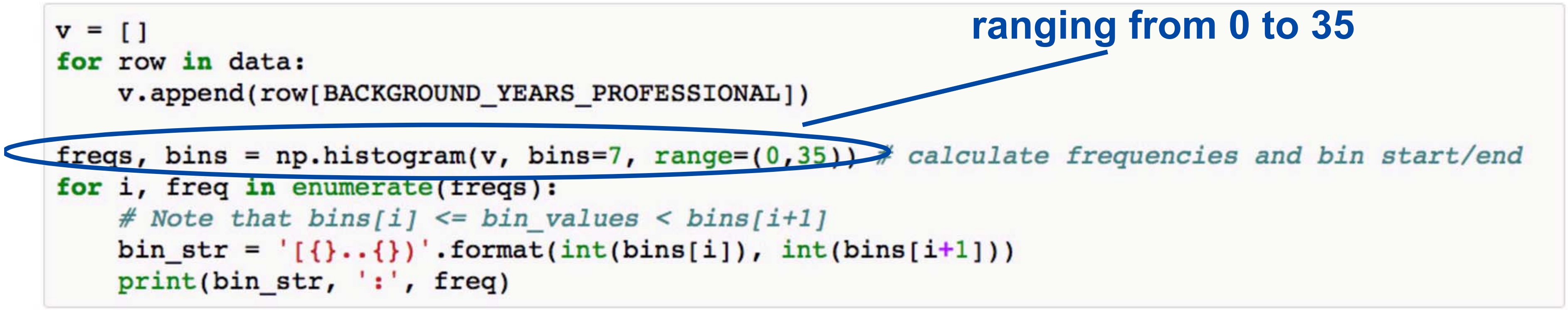


```
import numpy as np
for column_key in [BACKGROUND_YEARS_PROFESSIONAL, BACKGROUND_YEARS_PROGRAMMING]:
    v = [row[column_key] for row in data] # grab values
    print(column_key.upper())
    print("* Min..Max: {}..{}".format(np.nanmin(v), np.nanmax(v)))
    print("* Range: {}".format(np.nanmax(v)-np.nanmin(v)))
    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))
```

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



```python
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)
```

Jupyter Exercise : Cleaning, Statistics and Histograms

- Clean data
  - Cleaning float data
  - Cleaning timestamp data
- Statistics and histograms
  - Statistics with numpy
  - Binning and histograms
  - Which statistics best convey dispersion of experience data?
- More histograms
  - calculate histogram of programming experience
  - calculate histogram with bin size of 2 (EXTRA)

Visualisation

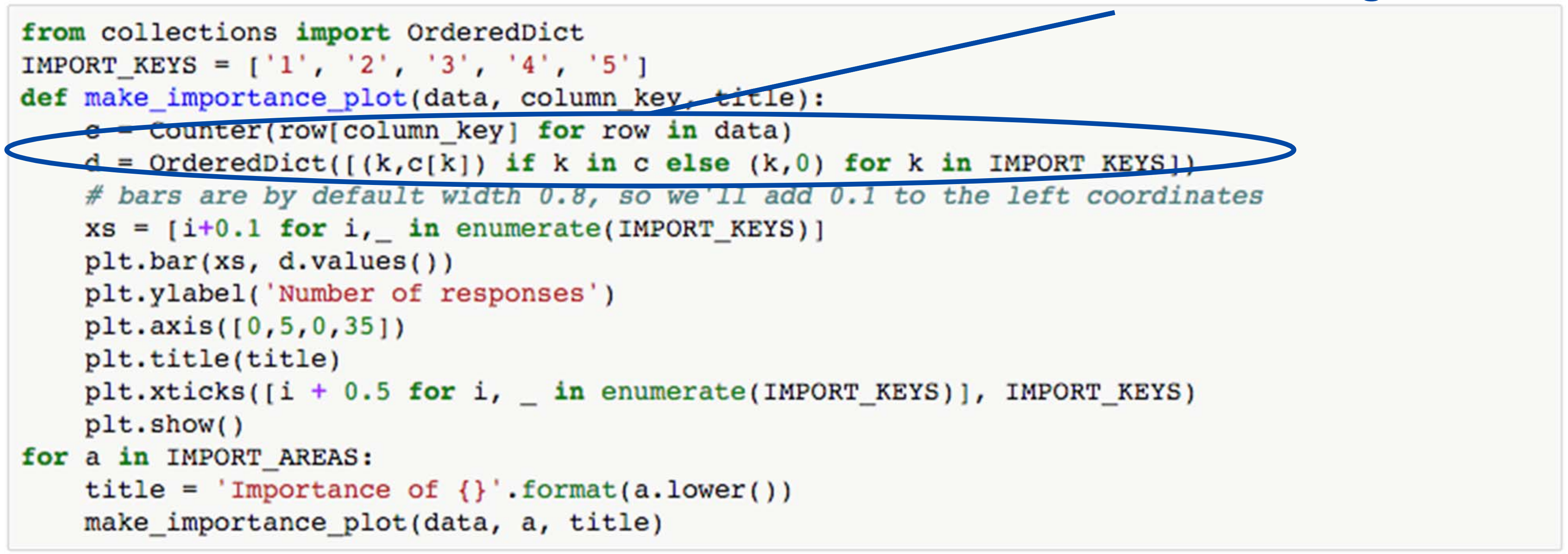
Visualising data with matplotlib

- 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

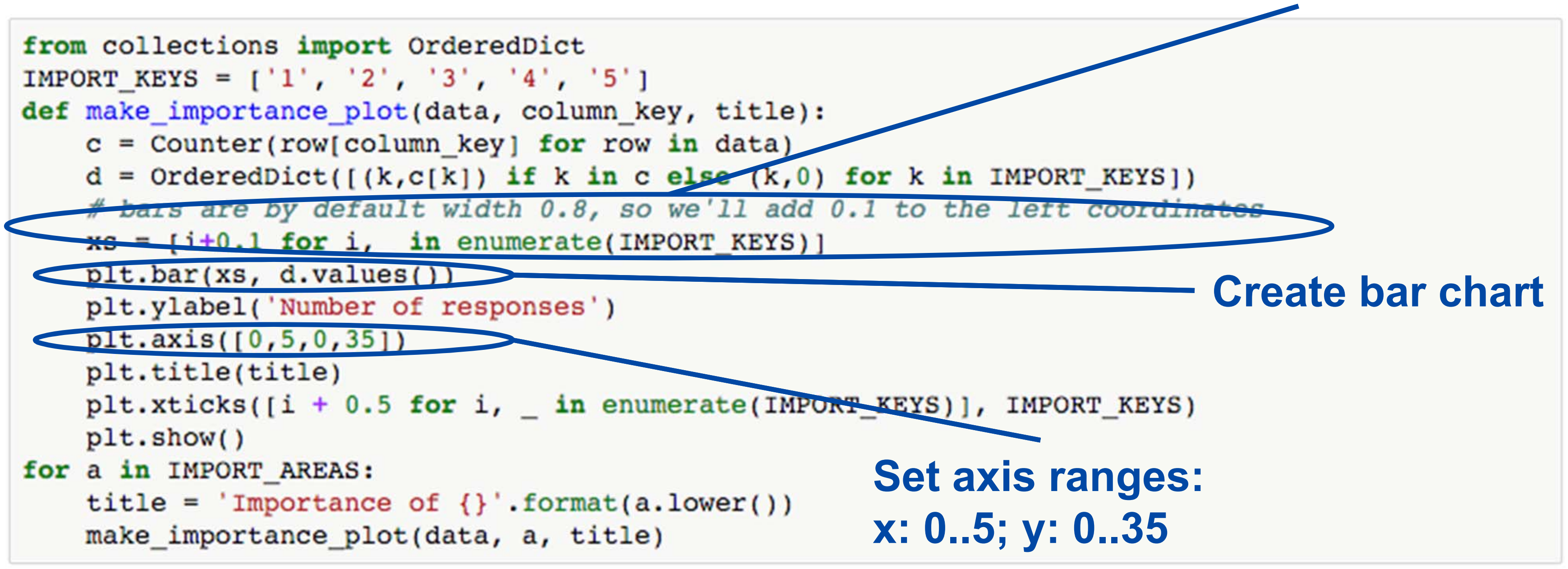
Use Counter to get frequency distribution
Reorder according to IMPORT_KEYS
Add default count of 0 for missing values



```python
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

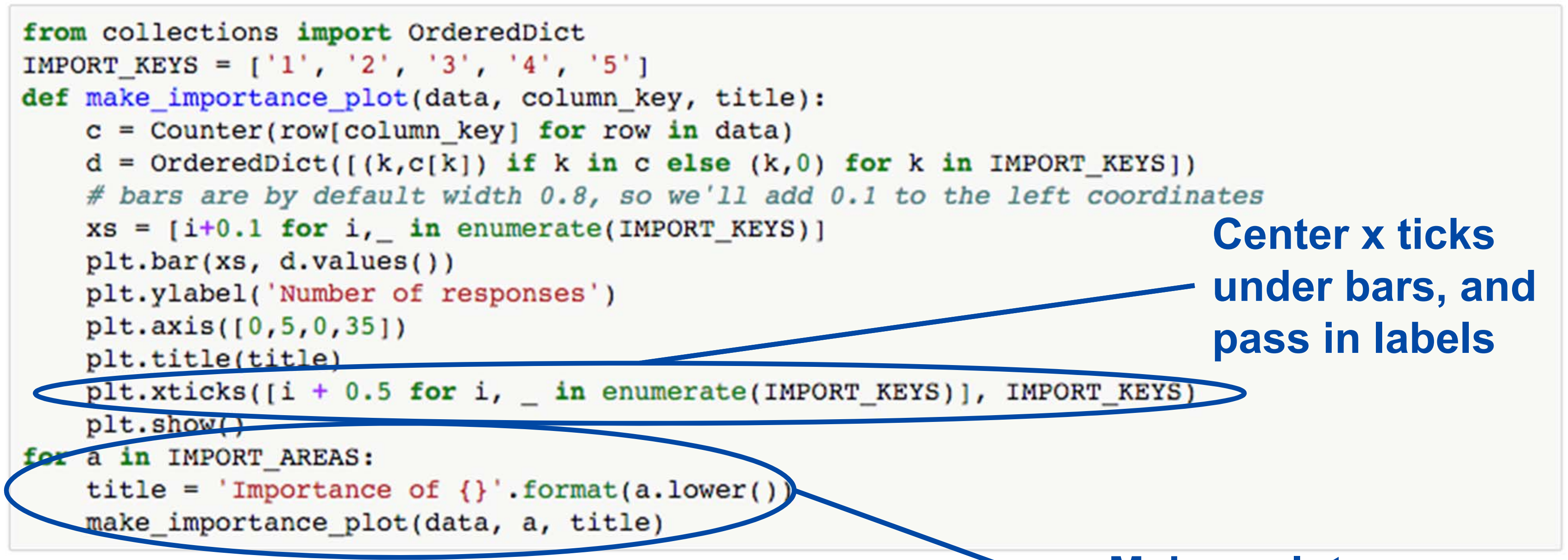


```
from collections import OrderedDict
IMPORT_KEYS = ['1', '2', '3', '4', '5']
def make_importance_plot(data, column_key, title):
    c = Counter(row[column_key] for row in data)
    d = OrderedDict([(k,c[k]) if k in c else (k,0) for k in IMPORT_KEYS])
    # bars are by default width 0.8, so we'll add 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

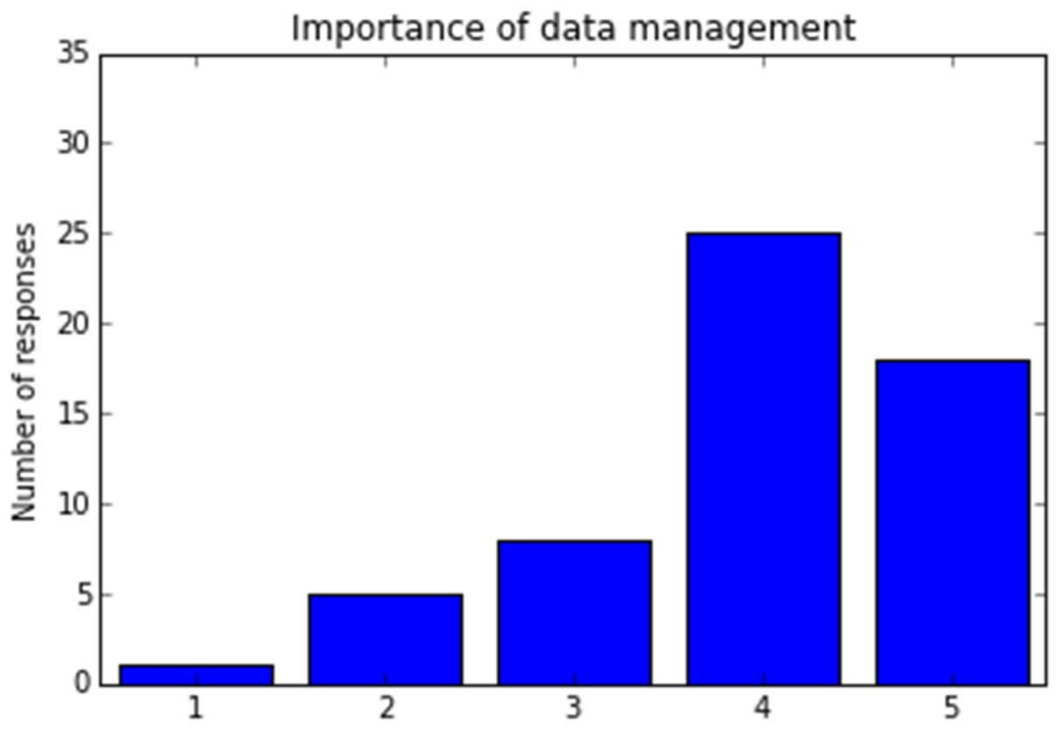


```
from collections import OrderedDict
IMPORT_KEYS = ['1', '2', '3', '4', '5']
def make_importance_plot(data, column_key, title):
    c = Counter(row[column_key] for row in data)
    d = OrderedDict([(k,c[k]) if k in c else (k,0) for k in IMPORT_KEYS])
    # bars are by default width 0.8, so we'll add 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

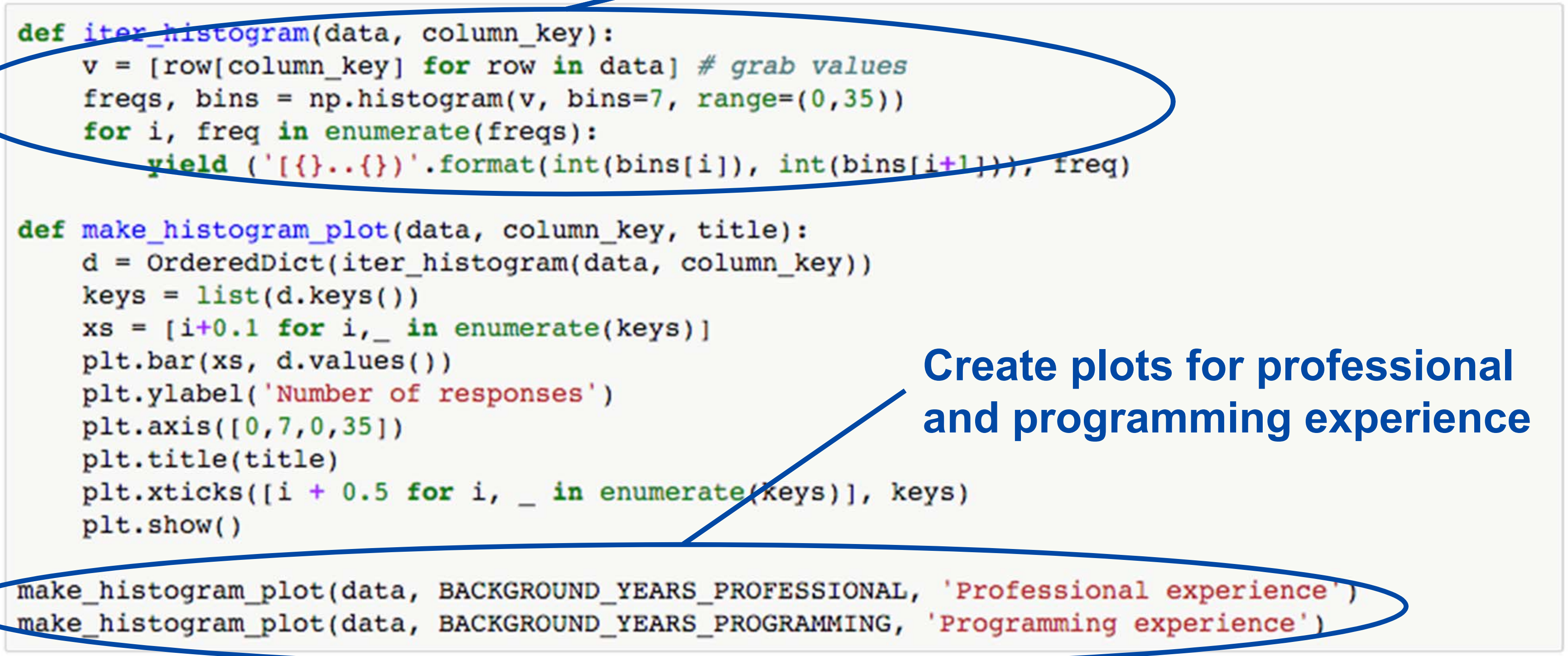
Make a plot for each area

A bar chart for data management ratings



Plotting a histogram

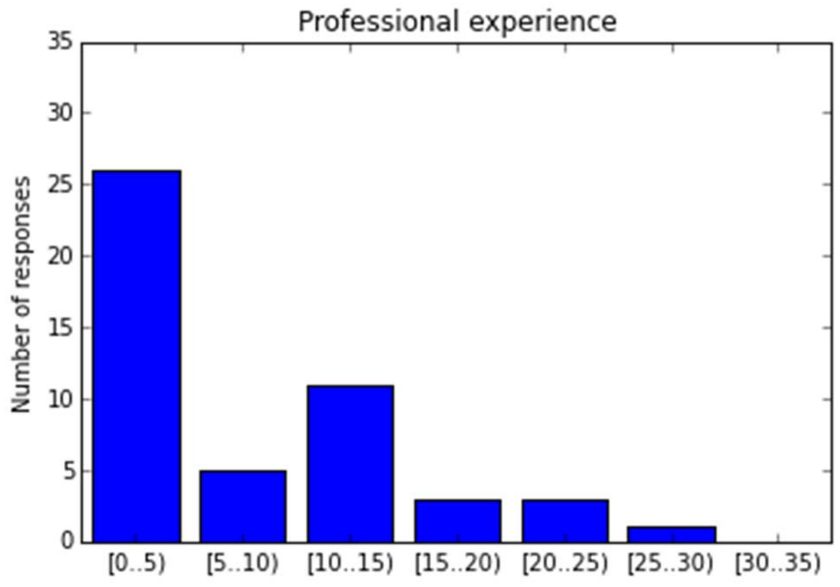
Yield bin_label, frequency pairs

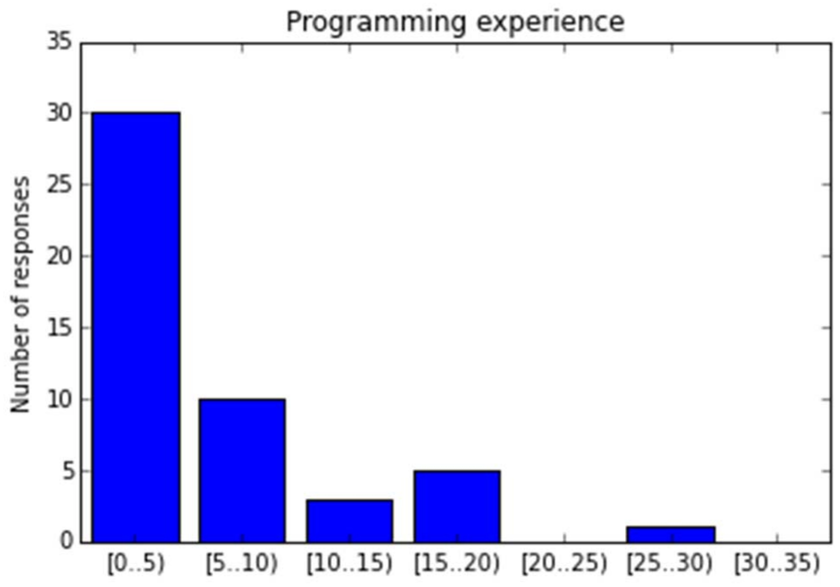


```python
def iter_histogram(data, column_key):
    v = [row[column_key] for row in data] # grab values
    freqs, bins = np.histogram(v, bins=7, range=(0,35))
    for i, freq in enumerate(freqs):
        yield ('[{}..{})'.format(int(bins[i]), int(bins[i+1])), freq)

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')
```

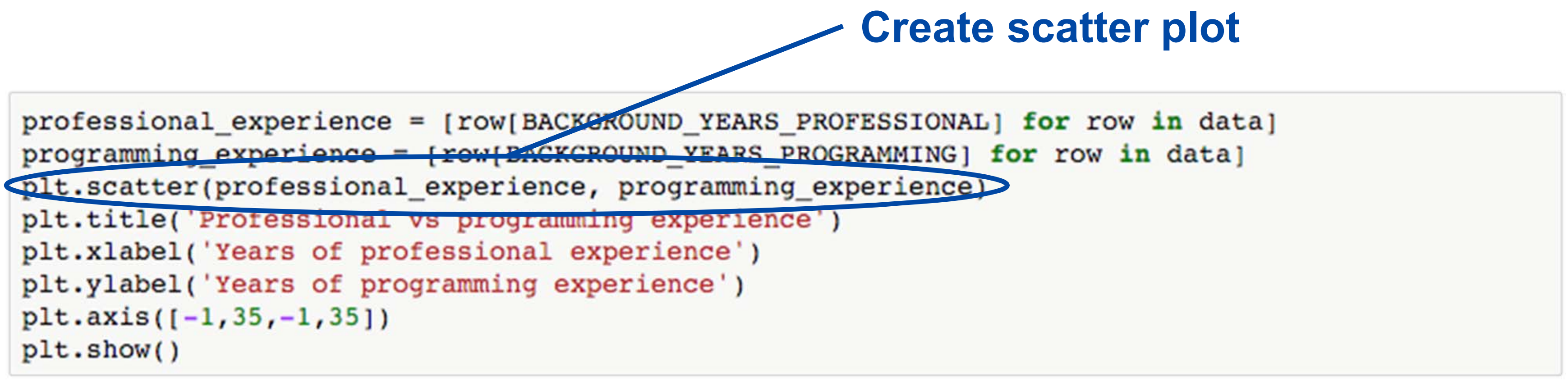
Create plots for professional and programming experience

Histograms for experience (bin size 5 years)



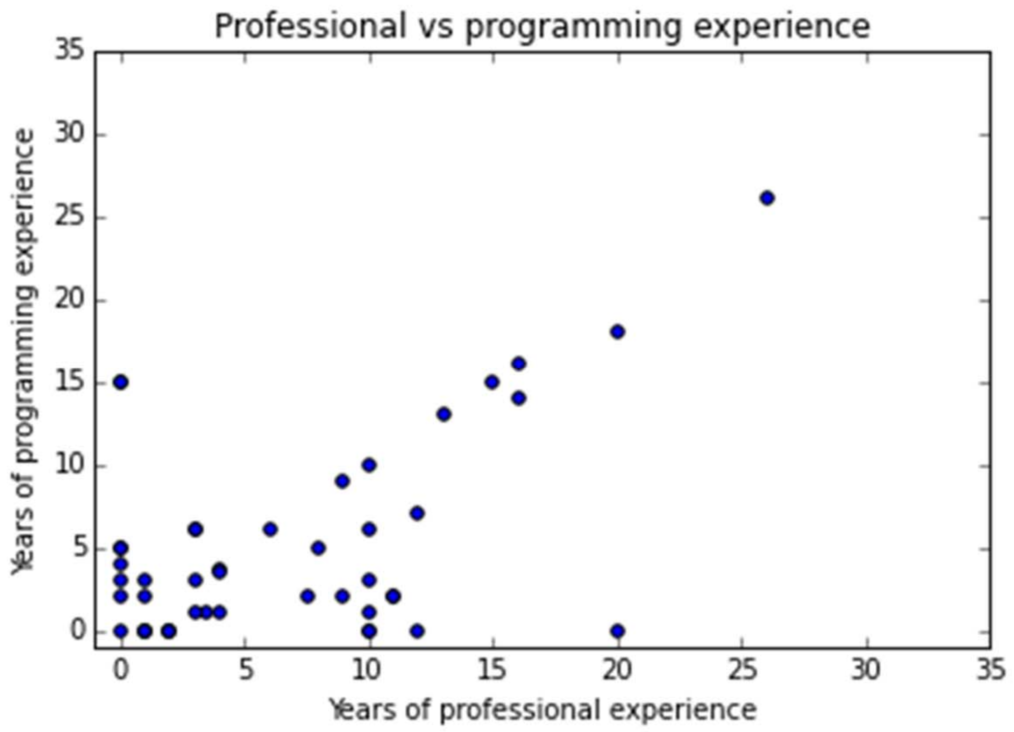


Creating a scatter plot



```
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



Jupyter Exercise: Visualisation with Matplotlib

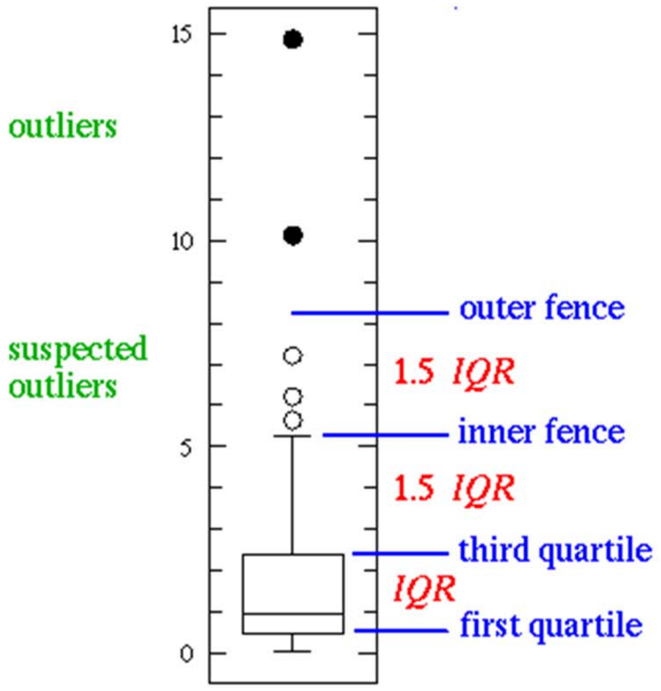
- Bar charts
  - Making a bar chart
  - create bar charts of know and future industries
- Histogram
  - Making a histogram
  - Are mean and standard deviation useful summary statistics here?
- Scatter plots
  - Making a scatterplot

Box plots and correlation

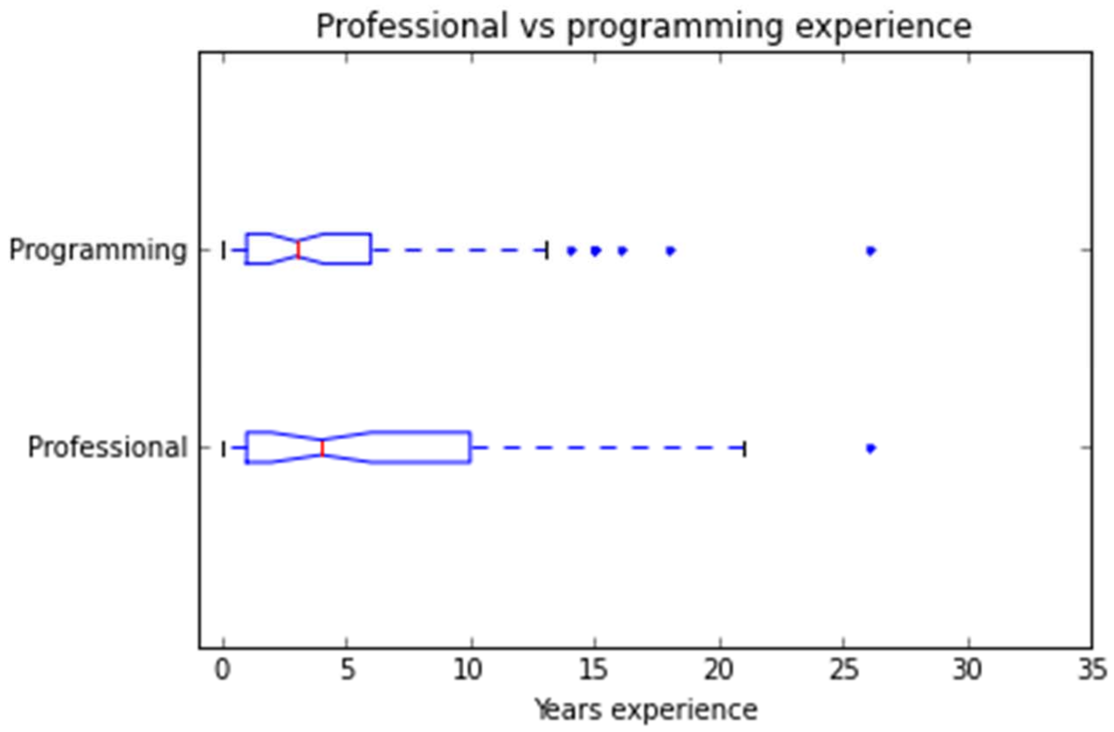
Using boxplots to compare distributions

- 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

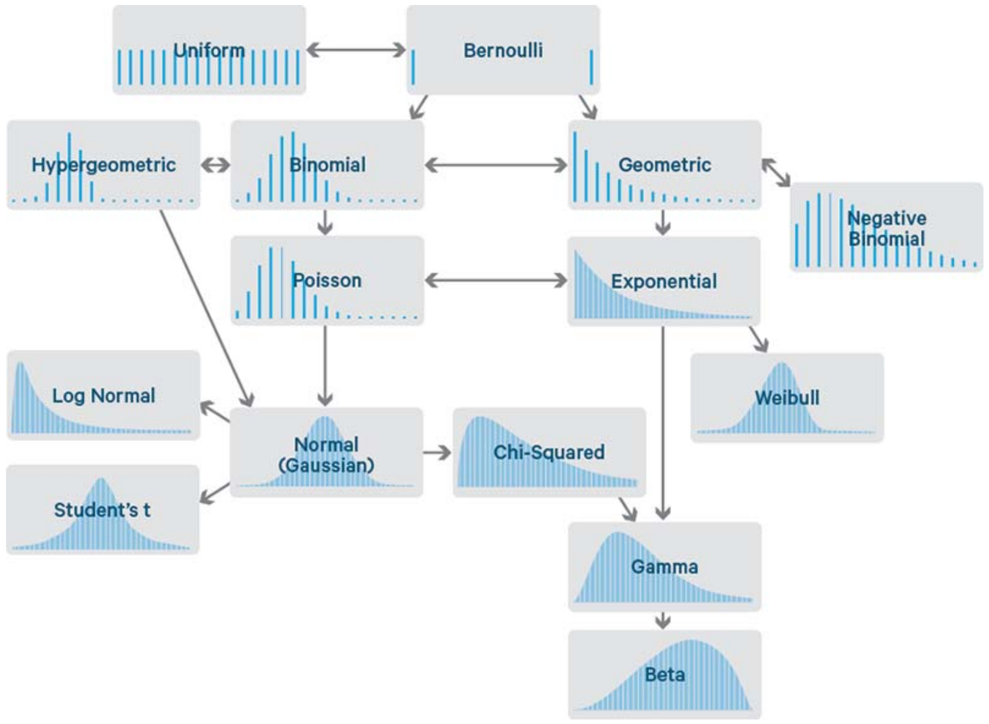


Using correlation statistics to measure dependence

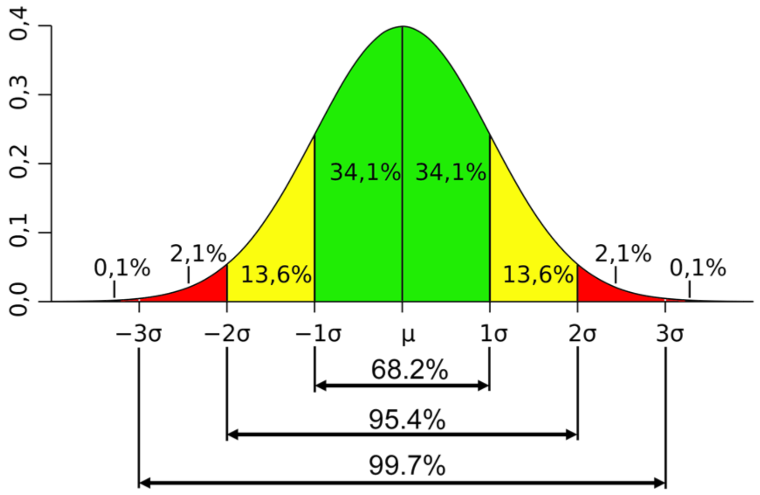
- Scipy includes various correlation statistics
  - Parametric test
    - Pearson's *r* for two normally distributed variables
  - Non-Parametric test
    - Spearman's *rho* for ratio data, ordinal data, etc   (rank-order correlation)
    - Kendall's *tau* for ordinal variables

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

Types of Data Distribution



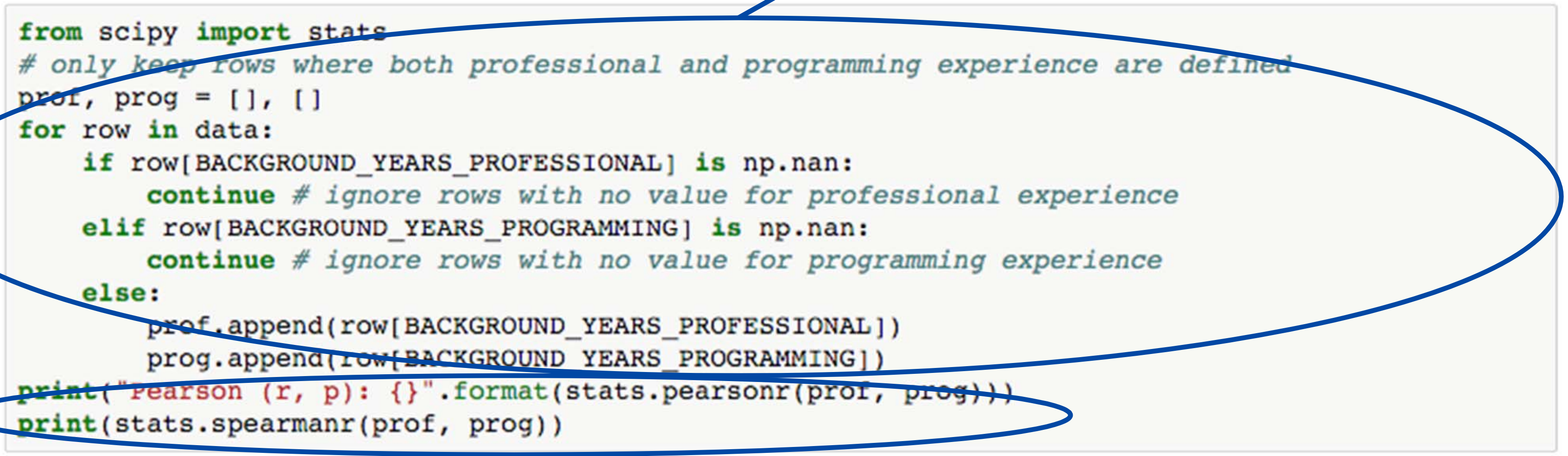
Normal (Gaussian) Distribution



https://blog.cloudera.com/blog/2015/12/common-probability-distributions-the-data-scientists-crib-sheet/

Calculating correlation

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



```python
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

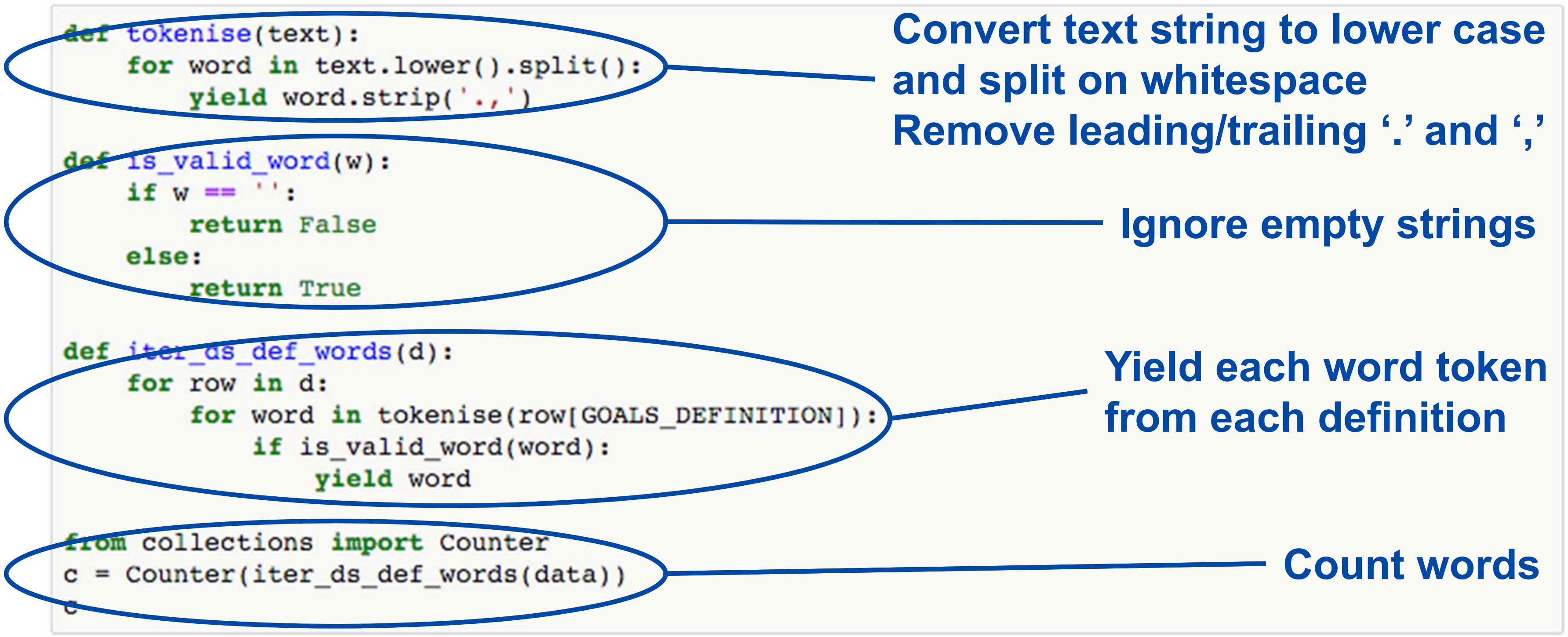
EXERCISE 5: Box plots and correlation

- Box plots
  - Visualising distributions with box plots
  - Which experience variable has more outliers?
  - Which experience variable has a tighter distribution?
- Correlation
  - Calculating correlation between two variables
  - Complete code to calculate correlation between importance ratings

Text Data

THE UNIVERSITY OF SYDNEY

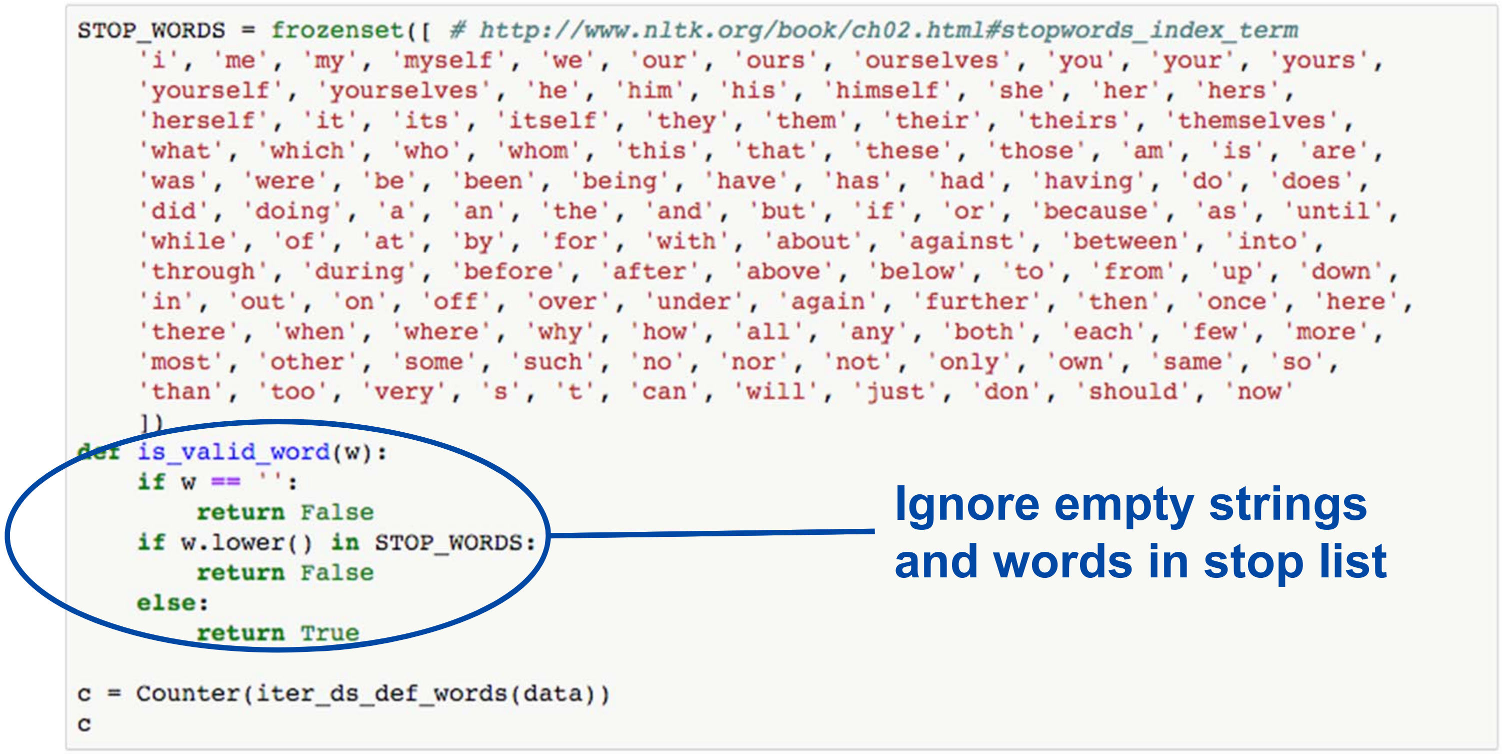
A simple whitespace tokeniser



```
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
```

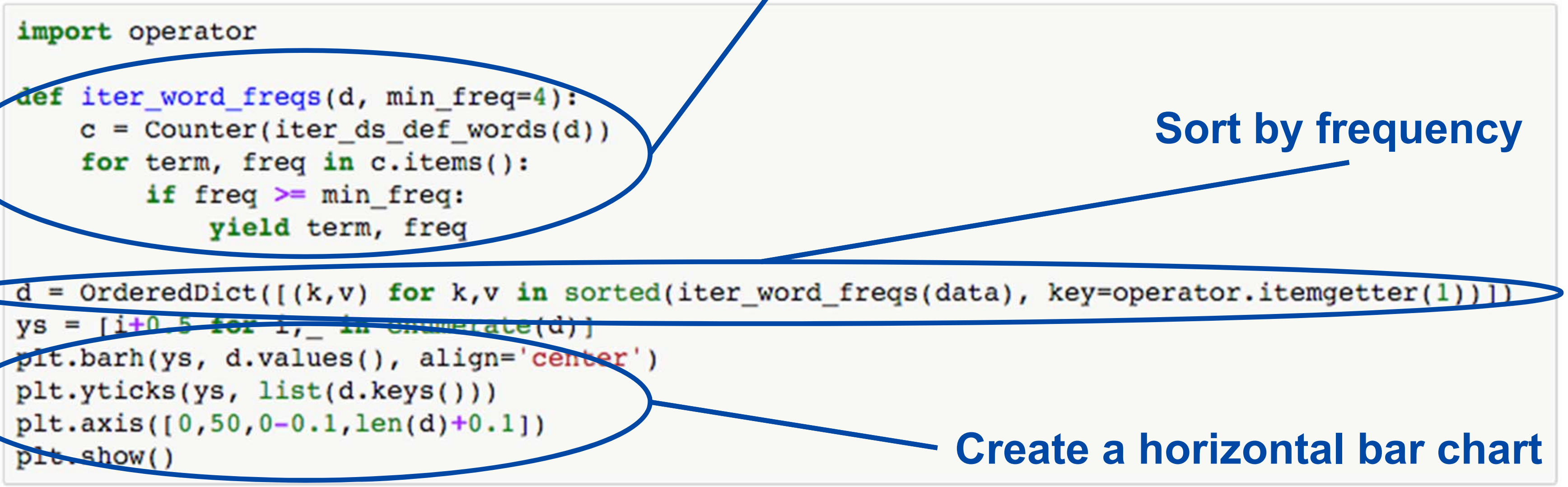
Ignore empty strings and words in stop list

Plotting most frequent words

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

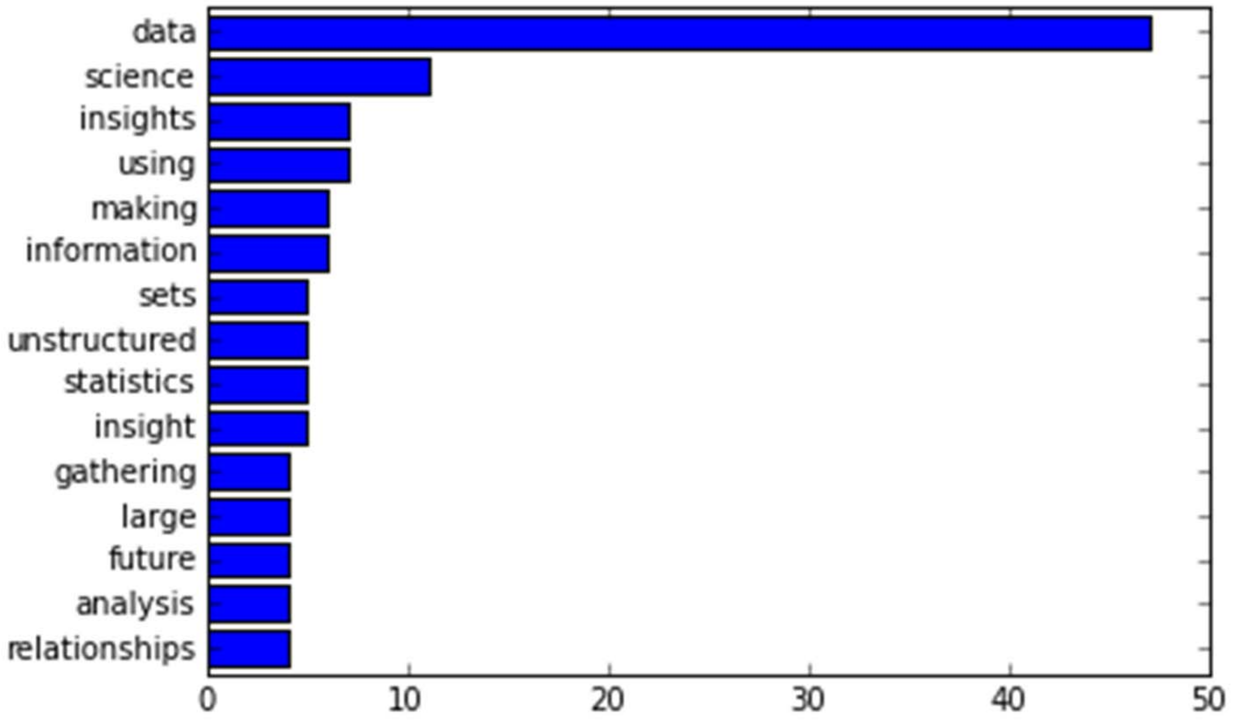
Sort by frequency

Create a horizontal bar chart



```python
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



EXERCISE 6: Calculating term frequencies

- Text data
  - Simple tokenisation and word counts
  - Removing stop words
  - Plotting term frequencies
  - What are the most important words?
  - How might we avoid repeated terms (e.g., insights, insight)?
  - Data science: using information; making insights? ☺

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
- Exercises went quite a bit into Python code already
  - Note: column-oriented data (arrays) would have made some processing easier than the row-oriented data read from CSV

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