COM6018 Data Science with Python

Lab 2: Reading and Writing Data Files

Jon Barker

In this lab

- Reading datasets from CSV files
- Prepare data for analysis
- Combining data from different dataset
- We will use pure Python (i.e., no 3rd party packages)…
- This will help us understand the benefits of the Pandas package!

Datasets vs Databases

Data scientists often talk about `dataset`; this should not be confused with a `database`.

- A `dataset` is a structured collection of data, typically stored in a single file.
- A `database` is a broader concept, typically a collection of datasets, stored in a database management system (DBMS).

Used for different purposes:

- A database will typically be designed to allow efficient querying, i.e. recalling information, cross-referencing items, etc.
- Data scientists are more often interested from learning something by using the whole dataset.

e.g. compare 'what was the weather yesterday?' with 'what will the weather be tomorrow?'

Structure of a Dataset

All datasets have a common structure:

- A dataset is a collection of records (or data items)
- Each record is a collection of fields (also called attributes or features)

For example, consider a spreadsheet of student data:

- Each row represents a student (a single record)
- For every row, there are the same set of columns (fields), e.g., name, degree title, date of birth, etc.

Note, the fields often have simple types (e.g., a string) but could have more complex type (e.g., a list or a set).

Using a dataset

- The data we used will typically be provided to us as a dataset stored in a file (or files).
    - Note though, that the data may originally have come from some sort of sensor, or from human input, or from a larger database.
- We will need to read the data from the file into our programs.
- We will then need to process the data in some way, e.g., to extract information, to find patterns, to make predictions, etc.
- We may then need to write the results of our processing back to a file.
    - Perhaps as a new dataset that will be used by another program or another group of people.
- So, understanding how to load and save dataset is a key skill for a data scientist.

Human-readable vs Machine-readable

Human-readable formats are designed to be read by humans

- Typically text based, e.g., ASCII or Unicode text files.
- Can be loaded into a text editor or printed to a terminal.
- Downside: not very compact, not very fast to load.
- CSV, TSV, JSON, XML, YAML
- Suitable for small or medium sized datasets.

Human-readable vs Machine-readable

Machine-readable formats are designed to be read by computers

- Typically binary files, i.e. not text files, close to the internal representation of the data.
- Often very compact, and fast to load.
- Downside: not human readable, so hard to check/debug.
- Examples: Avro, HDF5, Parquet
- Suitable for large datasets.

Data Formats using in COM6018

In this module we will be primarily using two human-readable formats:

- CSV (Comma Separated Values)
- JSON (JavaScript Object Notation)

We may also encounter a few other formats for specific tasks, e.g. YAML for configuration files, specific binary formats for storing image data, etc.

CSV (Comma Separated Values)

A simple text-based format for storing tabular data.

- Each record is stored on a separate line.
- Each field is separated by a comma.
- The first line may contain the names of the fields.

Example, a CSV file containing information about wind farms:

```
"id", "turbines", "height", "power"
"WF1355", 13, 53, 19500
"WF1364", 3, 60, 8250
"WF1356", 12, 60, 24000
"WF1357", 36, 60, 72000
```

TSV (Tab Separated Values) is similar, but uses tabs instead of commas.

Reading with csv.reader

csv files cab be read with the `reader` function from the `csv` module.

```python
import csv

# Open the file
with open('data/windfarm.csv') as csvfile:

    # Attach a reader to the file
    windfarm_reader = csv.reader(csvfile, skipinitialspace=True)

    # Read each row (as a list) and store them as a list of lists.
    data = [row for row in windfarm_reader]
```

Reading with csv.reader

The code on the previous slide will return a list of lists

```
[
 ['id', 'turbines', 'height', 'power'],
 ['WF1355', '13', '53', '19500.0'],
 ['WF1364', '3', '60', '8250.0'],
 ['WF1356', '12', '60', '24000.0'],
 ['WF1357', '36', '60', '72000.0']
]
```

This can be difficult to work with, as the data is not in a convenient format.

There is a better approach...

Reading with csv.DictReader

The same file can be read with the `DictReader` function from the `csv` module.

~~~
```python
import csv

with open('data/windfarm.csv') as csvfile:

    # Attach a DictReader to the file
    windfarm_reader = csv.DictReader(csvfile, skipinitialspace=True)

    # Read each row as a dictionary and store as a list of dictionaries.
    data = [row for row in windfarm_reader]
~~~

Reading with csv.DictReader

The data will now be returned as a list of dictionaries,

```
[
    {
        "id": 1355,
        "turbines": 13,
        "height": 53,
        "power": 19500
    },
    {
        "id": 1364,
        "turbines": 3,
        "height": 60,
        "power": 8250
    },
    // etc
]
```

Using the data

Working with a list of dictionaries is much easier than working with a list of lists.

For example, using a list of lists,

```python
# Get the height of the first windfarm

data[0][2]  #  not very clear!!
```

vs list of dictionaries,

```python
# Get the height of the first windfarm

data[0]['height']
```

Today's Lab

If you have clone the module GitHub repository then you should see,

```
materials/labs/
├── 000_Introduction.md
├── 010_python_intro.ipynb
├── 020_reading_data.ipynb
├── data
│   ├── Mine_Dataset.xls
│   ├── ch4.csv
│   ├── co2.csv
```

The lab is `020_reading_data.ipynb` and it will need the files `data/ch4.csv` and `data/co2.csv`.

Or you can download the notebook and data via links on Blackboard. (Click on 'Learning Materials' then 'Week 2')