Introduction to Python for Social Science

Lecture 2 - Data Structures and Pandas I

Musashi Harukawa, DPIR

2nd Week Hilary 2020

Overview

Last Week

- What is Python, and what can I use it for?
- What tools do I have to write, test and run Python code?
  - Opening up JupyterLab/Notebooks/IPython
  - Executing code in Jupyter, etc.
- Writing your first Python script and notebook.
  - Data Types
  - Data Structures

This Week

This week we will learn about a key library for working with **data**:

- Thinking about (tabular) data
- I/O
- Slicing, indexing
- Summarising

The Hard Truth About Data Science...

- Analysis usually takes $<30\%$ of your time.
- $>50\%$ of your time will be spent reading, cleaning, checking, storing, and cursing your data.
- Data cleaning is meticulous work, but that doesn't mean you can't be efficient.

Methodological Motivation and Background

Thinking About Data

We can think of a data point as having two properties:

1. Value
2. Relation (to other values)

- Inherent in value is an abstract type that the value is an instance of.
- In data science, we are often more concerned with the latter than the former for the bulk of the analysis. I say this because we don't tend to care about the exact values, beyond type, until the end, when we look at the output of our model. When building our pipeline, what matters most is the type and structure of data.

Data Structures

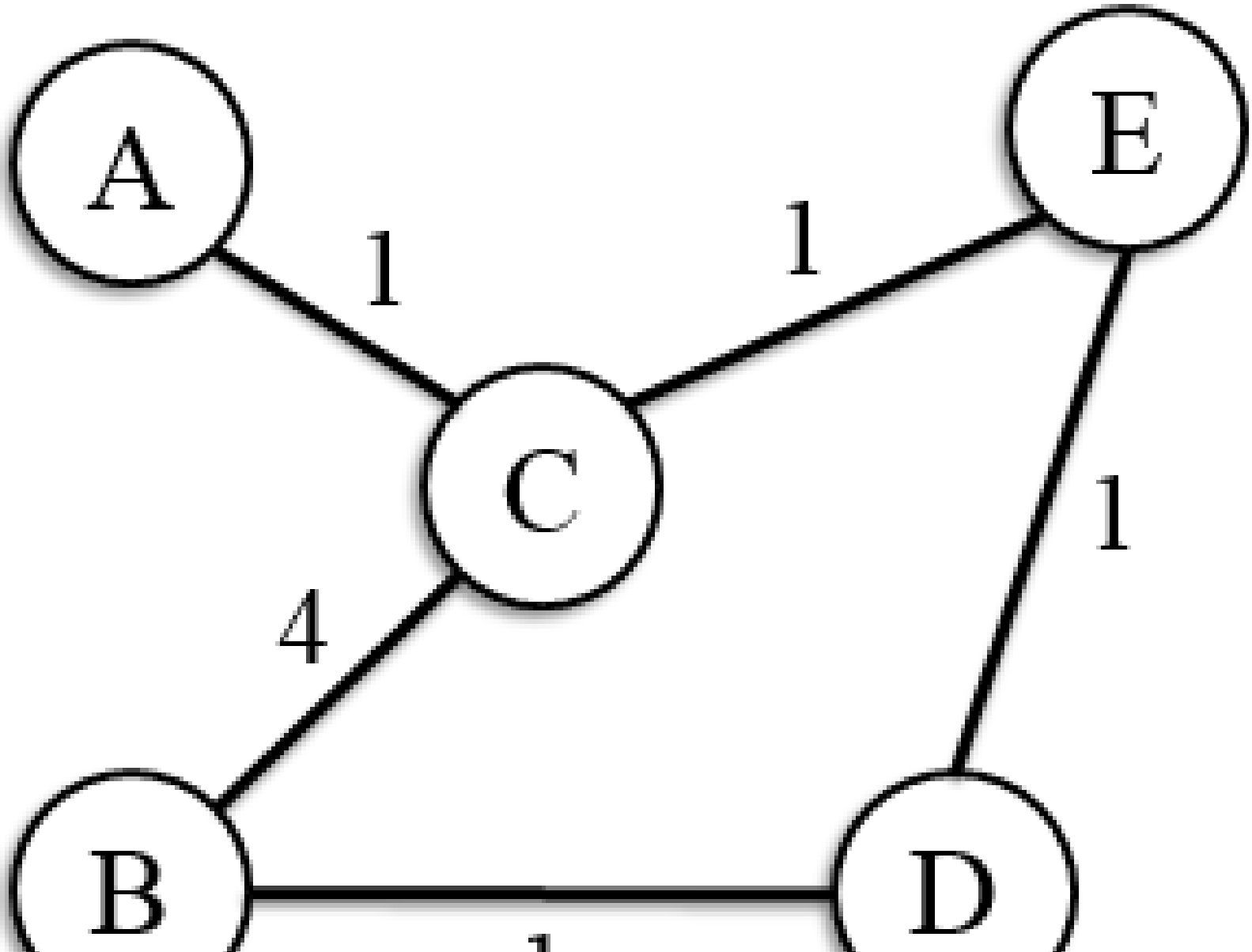
Three Ways of Structuring Data:

- Graph
- Tree (Hierarchical)
- **Tabular**

Option: Do an example of how we can structure data about the members of the class.

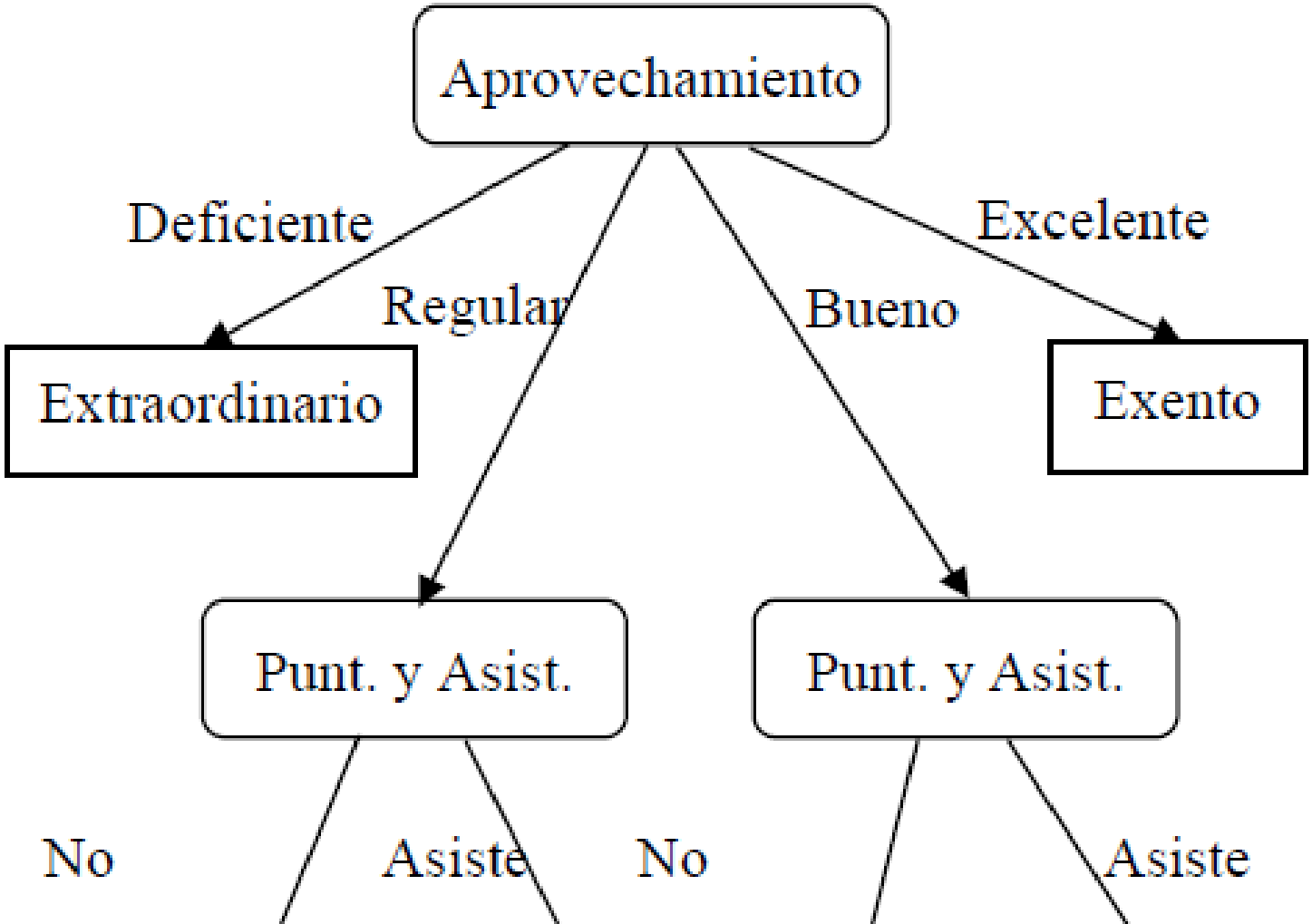
Graph

- Graphs are constructed of nodes (vertices) and edges.



Trees

- A tree is a type of graph with a number of properties
- Commonly used to represent hierarchical data structures, such as nested sets or dependency/flow diagrams.



Tabular Data

- Tabular data consists of an ordered arrangement of rows and columns.
- A common example is a spreadsheet.
- In this class will often be referred to as a matrix $\mathbf{X_{ij}}$ of i observations of j variables.
  - Note all columns must be of equal length (i), and all rows must be of equal length (j).

In this class we are primarily focused on tabular data. If your data is not tabular, you may want to figure out some way to coerce it to a tabular format because most statistical/ML models assume tabular data.

Coercing Data Structures

- In reality, processes we observe are rarely tabular.
  - Some observations may have special characteristics that others do not. (j is not equal for all i)
  - There may no be inherent ordering in our observations, or the characteristics (i or j are not orderable).
  - Observations may be related, overlapping, or nested in a way that is relevant to our model but not suitable for a two-dimensional table.
- *Take away*:
  - When we receive or generate tabular data, we should keep in mind the *data-generating process* and decide whether we are systematically losing information that is relevant to our model.
  - Trade-offs and subjective decisions are always necessary; make these clear and do your best to justify them.

Implementation

Data Formats

- Data analysis is usually done *in-memory*.
- We use a variety of *data formats* to store information *on disk*.
- The representation of data in these two mediums is different, but most data analysis software provides methods for reading in and writing out data.
- This process is not entirely trivial; different formats use different data types. Some potential pitfalls:
  - Incompatibility: data in one format may not work with some software.
  - Silent casting: data may change in unpredictable ways when converted between formats and programs.

On-disk Formats

There are a number of data storage formats that you should be aware of:

Format	Structure	Built-in Types	Human-Readable	Compatibil
`csv`	Tab	No	Yes	Any
`json`	Hier/Ord	Yes	Yes	Any
`xls(x)`	Tab/Rel	Yes	No	Excel
`dta`	Tabular	Yes	No	STATA
`HDF5`	Hier/Tab	Yes	No	C, Java, Pyth
`Feather`	Tab	Yes	No	Python, R

Human-Readable Formats

`csv` and `json`

- `csv` ("comma separated-values") is an extremely common tabular data storage format.
- Values are *delineated* by a special character, usually a comma.
  - There are reasons to not use commas, especially when working with text data.
- *Has no built-in data types*; this needs to be inferred by the parser.
- `json` (JavaScript Object Notation) is also extremely common, especially when using web data.
- Stores information as a mixture of key-value pairs and arrays (think dicts of lists).
- Working with `json` usually requires us to coerce hierarchical data to tabular data.

Closed-source Binary Formats

`xls(x)` and `dta`

- `xls(x)` is the format used by Microsoft Excel
- If there is only one sheet, then this is tabular and essentially equivalent to `csv`.
- If there are multiple sheets, `pandas` represents it as a list of tabular data frames.
- `dta` is the format used by STATA.
- More common in social sciences, but essentially unheard of in professional contexts.
- May be preferred for compatibility with STATA, otherwise would not recommend.

Data Science-Specific Formats

`HDF5` and `Feather`

- `HDF5` is a commonly-used data storage format in data science, but not in academia.
- Has a lot of nice properties, including efficient compression and fast reading.
- `Feather` was created by Hadley Wickham and Wes McKinney as a fast, consistent and convenient data storage format for cross-usage between R and Python.
- I highly recommend this if you work a lot with R and Python, or want to use both in the same project.

DBMS and SQL

- In industry, database management systems (DBMS) are used to store and query large quantities of data in a reliable way.
- SQL-compliant databases are a common type. SQL is a database managing and querying language.
- If your research requires you to constantly be collecting data and ensuring its reliability, you should opt for a DBMS instead of one of the filetypes mentioned above.

pandas

`pandas` is a very popular library for working with tabular data structures in Python. Before we start using it, let's go over some of the ways it can be useful to you as a social science researcher.

Advantages of pandas

- provides fast, flexible data structures
- extensive array of convenient functions
- compatible with most data science libraries and data types

When you should **not** use `pandas`

- Your data is not coercible to a tabular structure.
- When your dataset is too large to load in your computer's memory (or loading it uses most of your RAM).

- `pandas` is so standard, that it makes more sense to talk about the scenarios in which you would not want to use pandas
- Note on RAM: It's not straightforward to predict the size of a dataset loaded into `pandas`. There are also options for dealing with datasets this large, although those are beyond what I can discuss in the lecture.

Data I/O

pandas comes with functions for reading and writing to all kinds of data formats. A quick list can be viewed using tab completion:

```
In [1]: import pandas as pd

In [2]: pd.read_<TAB>
 read_clipboard() read_hdf()       read_sas()
 read_csv()       read_html()      read_sql()
 read_excel()     read_json()      read_sql_query()
 read_feather()   read_msgpack()   read_sql_table()
 read_fwf()       read_parquet()   read_stata()
 read_gbq()       read_pickle()    read_table
```

- Explain what I/O means.
- This is a good moment to mention tab completion. Revisit it later during the coding tutorial.

Coding Tutorial

Today, we learn about the following in `pandas`:

- Data I/O
- Understanding the `DataFrame` and `Series`
- Indexing and Slicing
- First look at the data
- Summary functions

Data I/O

- As seen earlier, pandas has methods for reading in data from various formats.
- A strength of learning a library like `pandas` is that we can analyse tabular data regardless of the source format.

Understanding `DataFrame` and `Series`

- `pandas` contains two native data containers:
  - `pandas.DataFrame`: A two-dimensional* labelled matrix
  - `pandas.Series`: A one-dimensional labelled array

Indexing and Slicing

- The most fundamental action in data analysis is the ability to "select" elements within your dataset.
- With tabular data, we usually want to select some rows, some columns, or specific cells.
- `pandas` data frames have explicit (named) row- and column-indices, as well as implicit indices because they all elements are ordered and named. We will learn methods for leveraging both.

First Look at the Data

- When working with data, your first step should always be *getting to know the data*. Ask questions like:
  - What are the dimensions of the dataset?
  - What information is contained in the columns? in the rows?
  - How is my data organised? (Long/tidy vs. wide format)
  - What data types are each of the columns? Is this expected?
  - How sparse is my data? (Looking for NAs)
- There's nothing worse than trying to debug code that's taken hours to write only to discover that the problem lies in your data!

Summary Functions

- Part of the function and appeal of data analysis is to reduce millions of data points to a few summary numbers that capture key information that you are looking for.
- Summary functions do this: they summarise a large number of observations to one or a few values that tell you what you need to know. They are also known as *statistics*.
- Basic examples include mean, sum, variance, skew, but also more advanced statistics such as regression coefficients, Kolmogorov-Smirnov tests, and even the output of machine learning algorithms!