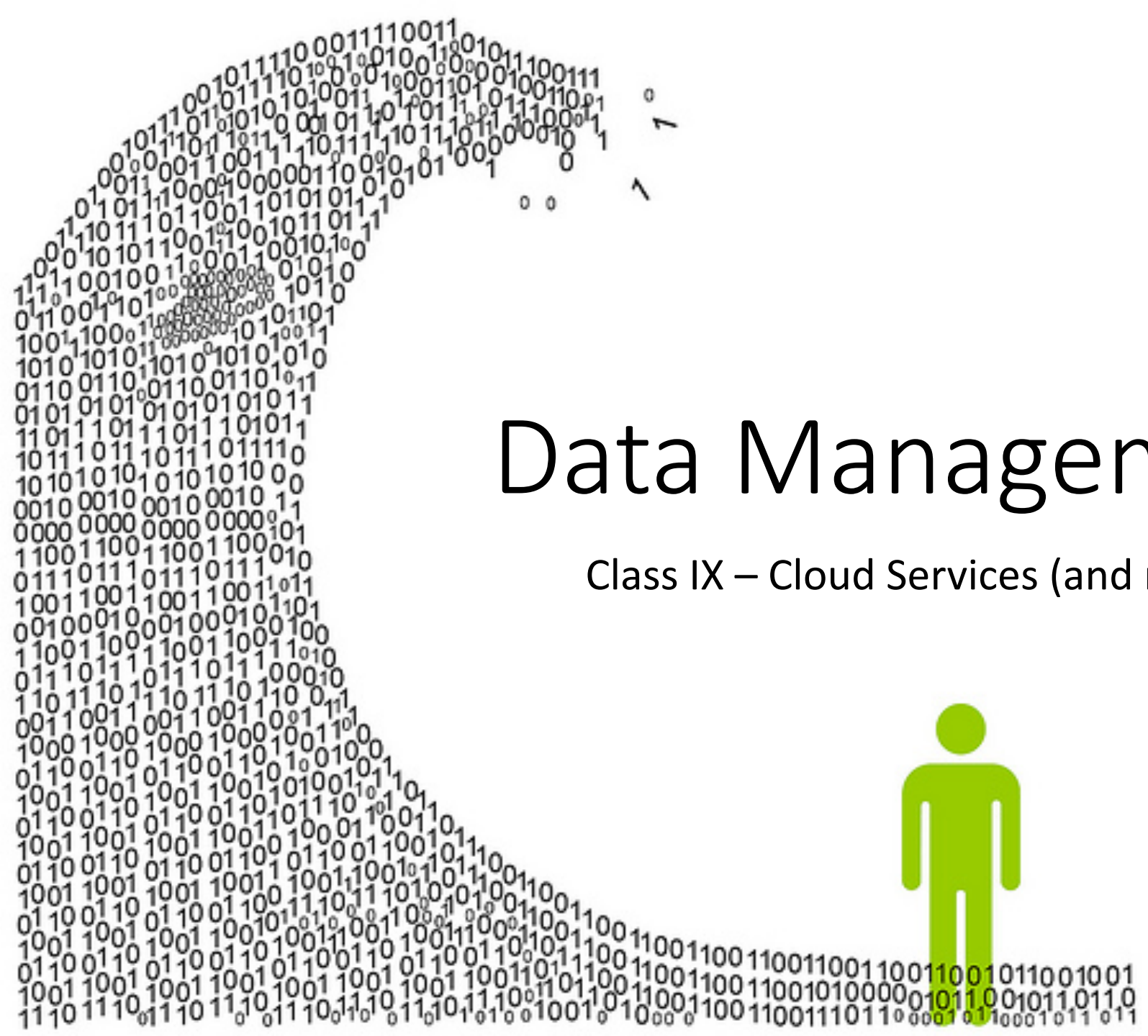
Data Management

Class IX – Cloud Services (and more)



Revision

- We have now learned how to use two very different types of database – SQL and NoSQL. We've imported our own research data into each of them and run various kinds of search queries.
- We've also seen how to use indexing to speed up database operations, and how to stream data so we can analyse data sets that are too big for the computer's memory.

Today's Class

- This is the second-to-last class, and I have two objectives…
  - Firstly, I want to cover a few topics which have come up over the past week and I think it would be good to revise – especially with regard to tips and tricks for successfully importing data.
  - Secondly, I want to introduce you very briefly to how Cloud services – and especially databases in the Cloud – can be used in research.

Tips for Data Import
1) Understanding Errors

- Programming error messages are honestly one of the worst things about doing data science. Everything else about using computers is slowly getting easier and more user-friendly; error messages are still stuck in the 1980s.
- However, trying to understand what an error message is saying to you is the quickest way to get good at fixing, or “debugging”, your programs and scripts.
- Don’t just say “it’s an error!” and panic; by learning some of the most common error messages, you can start to quickly figure out what’s causing your problems.

Error Messages in Python

- Python error messages are often huge – pages and pages of weird-looking information. The first thing to know is where the really useful bits in an error message are located:
  - The top of the error message should include an arrow pointing to the line of you code where the error occurred;
  - The bottom of the error message is an actual description of the kind of error that occurred.

  - All the stuff in between those lines can usually be safely ignored!

Common Python Error Messages

- **`NameError: name ‘~~~’ is not defined`**
  - This is probably the most common error you’ll encounter in Python. Luckily it means something very simple… You’ve tried to do something to an object that doesn’t actually exist. 99% of the time, that means you’ve mis-spelled the name of an object, or just used the wrong name for it.

- **`ModuleNotFoundError: No module named ‘~~~’`**
  - This is also pretty simple – it usually happens when you try to **import** a library, and it means that library isn’t installed on your system. Go to a terminal window and install it using **conda** or **pip**, then try again.

Common Python Error Messages (2)

- **`TypeError:`**
  - This is probably the second most common type of error, and it basically means that you've tried to do something using the wrong "type" of variable. A variable's "type" is whether it's a string, an integer, etc.
  - This happens most often when you try to do mathematical operations but your numbers are actually strings, not integers; or when you have missing data ("NoneType") that your program doesn't know how to handle.

Common Python Error Messages (2)

- **`SyntaxError:`**
  - This error happens when your code has a mistake in it – the most common one is a missing bracket messing things up! Check carefully to make sure all your code is properly formatted; just one missing character can break everything.
- **ProgrammingError**:
  - When you're working with MySQL, a "ProgrammingError" usually means there's a syntax error not in your Python, but in your SQL code.

Other Errors

- There are a lot of other error messages you might encounter, and we can't cover them all here.
- The real trick to handling errors is…
  - First, **read** the last line of the error message. They're written in English, and while the language is very technical, it can give you good hints; if something says "ConnectionError", you can guess it's to do with your connection (to a database or a website, perhaps), right?
  - Second, **copy and paste the error into Google**. Just copy and paste the last line (*SomethingError: error description here)*, and see what comes up. I bet a lot of other people have come across the same problem before! Websites like Stack Overflow are full of great advice for dealing with coding problems.

Tips for Data Import
2) Fixing Text Encoding Problems

- One of the most common problems you'll encounter when you try to import data from any source – a file you downloaded, or received from a colleague, or even a file you created yourself – is text encoding.
- This creates error messages too, of course: usually **EncodingError** or **UnicodeError**.
- You might see a message like "**Unicode cannot decode byte x034 at position 4**", or something similar. This means that the file you're loading isn't one that Python can read as Unicode, and probably uses a different encoding.

Fixing Text Encoding

- Fortunately, almost every import command (like **read_csv()** or just the regular **open()** file command) allows you to add a parameter:
  - `pd.read_csv(“filename.csv”, encoding=“latin1”)`
- Unfortunately, figuring out what encoding to set in that parameter is often a matter of trial and error. “latin1” is a popular (but outdated) encoding for European languages, so it’s worth trying that first. “cp1252” is another old encoding commonly used for European texts.
- Japanese text is often encoded in “shift_jis”, and Chinese text in “big5”.

Tips for Data Import
3) Moving data to and from R

- A lot of data science projects involve doing work in both Python and R. Python is much more well-suited for some tasks – handling databases and cloud services, doing complex machine learning, and so on – while R has better libraries for many kinds of statistics.
- Moving data between Python and R can be tricky. A lot of you have tried using **pyreadr** – which reads R data files in Python – or **rpy2** – which lets you run R commands from Python – with limited success.
- The most failsafe way to do this is actually to use the **Feather** file format, which saves a Python or R data frame in a secure, open format that can be easily shared with other users or imported into a different program

Using Feather in R and Python

- In R, you need to install and import the **feather** library. Then you can save a dataframe to disc, or load one into R from disc, by using commands like:
  - `write_feather(my_dataframe, “my_filename.feather”)`
  - `my_dataframe <- read_feather(“my_filename.feather”)`

- In Python, install the **pyarrow** library, and you can load and save Feather files directly from pandas…
  - `my_dataframe.to_feather(“my_filename.feather”)`
  - `my_dataframe = pd.read_feather(“my_filename.feather”)`

A quick note on Feather…

- Feather is a really efficient and safe way to move data frames between R and Python, or to share data frames with your colleagues.
- However, Feather assumes that your data frames have a regular shape… In other words, if your data frame does something unusual like storing lists inside columns (breaking the “2D table” concept of a data frame), you can’t save it to Feather.
- In general, if you need to store lists or objects in a hierarchical way, you should be using *dictionaries* rather than data frames.

Cloud Services

“Just someone else’s computer”

- Yes, the Cloud is really just a huge number of powerful computers belonging to Google, Amazon, Microsoft and so on.
- However, that simple description ignores the huge benefit of storing and processing your data on “someone else’s computer”.
- Those computers are:
  - Much faster than your laptop;
  - More secure and regularly backed up than your laptop;
  - Have far more memory and storage than your laptop;
  - and are connected to the Internet at much higher speed than your WiFi connection.

Cloud Service Providers

- There are a lot of different Cloud Service providers out there. The three big ones are:
  - Amazon (Web Services, or AWS)
  - Google (Cloud Platform, GCP)
  - Microsoft (Azure)

- All of those companies offer basically the same kinds of cloud services – and there are many smaller companies, such as Digital Ocean, which also provide specific kinds of cloud service.
- My research teams use Google Cloud, but AWS and Azure are also good options.

What is a “Cloud Service”?

- Any company’s Cloud is made up of a host of different “services”. Services include things like:
  - Servers and ‘virtual machines’ where you can run code – doing analytics, running a website, etc.;
  - Data storage services like databases or long-term data archives;
  - API services giving access to advanced technologies like AI and machine learning, language translation, image or speech recognition, etc.
  - Support services like security, communications etc., which help to tie together all those other services and allow them to work together.

Databases in the Cloud

- What we're interested in in this class is the potential for storing, accessing and sharing data in the Cloud.
- There are two very different approaches to running a database in the Cloud…
  - You can run standard database software on a Cloud server – so for example, you could have MySQL or MongoDB running on a server provided by Amazon or Google. This would let all of your team access it from anywhere in the world.
  - You could use a native Cloud database. You'd never know or care about where or how your data is stored – just put data in and take it out, trusting the Cloud company to worry about all the rest.

When to use the Cloud

- Cloud services are a great option in certain circumstances.
  - If your data are simply too large to fit onto a standard laptop, huge Cloud databases can be a great solution to the problem.
  - If your data needs to be worked on by a lot of different people around the world, running a database in the Cloud is an effective solution.
  - If you have at least one team member who is very literate about IT and programming (maybe this could be you?), and can set up Cloud access, security etc. for everyone else, then using a Cloud service can be a great idea.
  - If you need to do something like running a web service (e.g. creating an interface for student RAs to work on tasks), Cloud services are ideal.

When **not** to use the Cloud

- The Cloud is a popular buzzword… but it's not always the answer to a research team's problems.
  - If you don't have someone who is a decent programmer and can get Cloud services working well for your team, trying to do so will just cause extra problems for you in the long run.
  - If your data are small enough to just share on Dropbox, you don't need the Cloud (well, Dropbox kind of *is* a Cloud service…).
  - If your data are extremely sensitive – e.g. commercial or private information – then you probably shouldn't put it in the Cloud unless you're *incredibly* confident about your IT security skills.