UNIVERSITÄT BERN

CAS Applied Data Science

Module 4

Best Practices for Data Science and Scientific Computing

2023-10-20

Objective / research / business case formulation

UNIVERSITÄT BERN

- Use research from well known credible sources.
- Define clear and measurable, objectives.
- Think through the purpose of the project
- Think of possible pitfalls of the project
- Think easy not complicated.

Write Programs for People, Not Computers

- ***a program should not require its readers to hold more than a handful of facts in memory at once (1a)***.
  - Programs are for people not computers.
  - Write coherent easy to understand code - not 100… of line of code.
  - Think scientifically.
- ***make names consistent, distinctive, and meaningful (1b)***
  - Keep naming conventions.
  - Stay consistent.
  - Use meaningful variable names.
- ***make code style and formatting consistent (1c)***
  - *Stick to a programming single format - Classes, Functions, OOP.*
  - *Indentation.*
  - *Program comments.*

Group 2

UNIVERSITÄT BERN

- Rigorous Mathematical Analysis before Data Analysis

e.g. – look for p-value

- Specialized Engineers should be hired to better

understand the scope of the project data.

Group 2

Ethics and BP in Data Cleaning and Preprocessing

- Transparency and documentation
- Bias and fairness
  - E.g., racial, gender, socio-economic-status …
  - Also cherry-picking
- Handle missing Data
  - Can be biased
- Privacy
  - Anonymization
- Communication
  - If data is biased / low quality, don’t overemphasize results
- Feedback-loop with collaborators

Make incremental changes

- Work in small steps with frequent feedback and course correction
  - real world changes frequently
  - working code after each iteration, iteration cycles 1 week
- Use a version control system
  - like Git & GitHub
  - compare and work on different versions
- Put everything that has been created manually in version control
  - include everything in VCS, code, metadata etc.
  - except (large) original Data, Pictures etc. → Archives

Group 4 - Feature Engineering

- reproducible, understandable → Transparency

  → able to understand what exactly has been done

- explain why you do each step

  → able to understand reasoning

Examples

- dataflow diagram of code
- normalisation of variables → explain why

Group 4 - Don’t repeat yourself

- Create programs for steps that repeatedly are used (loading data e.g.)
- Reuse instead of rewrite → saves time

- every piece of data must have a single authoritative representation in the system?????

Group 5: Plan for mistakes.

UNIVERSITÄT BERN

Mistakes are inevitable - don't panic!

(a) Add assertions to programs to check their operation.
   i. checkpoints with python command "assert": stops if result is not as expected
   ii. works also as a form of documentation

(b) Use an off-the-shelf unit testing library.
   i. "Automated testing": unit tests / integration tests / regression tests

(c) Turn bugs into test cases.
   i. Anticipate possible errors, program accordingly
   ii. Try to provoke errors to see if they still occur

(d) Use a symbolic debugger.
   i. is a line-by-line "interactive program inspector"
   ii. lets the programmer witness live what is happening in the code, e.g with the variables

Group 5

Group 6 : Optimize software only after it works correctly.

UNIVERSITÄT BERN

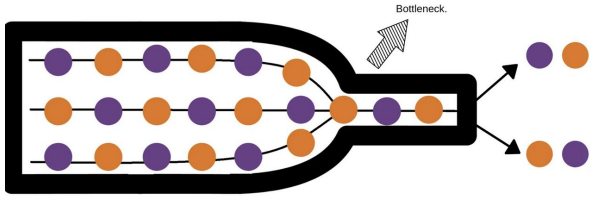
Goal : Improve the efficiency of a given code

1) Run a functional code

2) Determine the needs to optimise it more

   a) Use a profiler to identify bottlenecks

      i) Program analysis determining which lines in the code are taking more time and/or CPU ?



Group 6 : Optimize software only after it works correctly.

UNIVERSITÄT BERN

2 categories of coding languages:

- highest-level language (e.g. Python) : slower but more intuitive
- lowest-level languages (e.g. C++) : faster but more difficult to write

Recommendations

- **(b) Write code in the highest-level language possible.**
- Switch to a lowest-level language when the program performances require it

Group 7

Group 7 - Presentation and publication

UNIVERSITÄT BERN

Why?

- A good documentation makes the code reusable and lowers maintenance costs. Furthermore it allows a smooth handover.

How?

- It is not about inline documentation.
- Document interfaces and reasons, not implementations (explain inputs and outputs).
- If a substantial description of the implementation of a piece of software is needed, refactor code in preference to explaining how it works (if possible).
- Embed the documentation for a piece of software in that software and update it.

  -> Document generator examples: “Javadoc, Doxygen, or Sphinx”

  -> Alternative: “literate programming”, for example “knitr” or “IPython Notebooks”

Group 8

Group 8

b
UNIVERSITÄT
BERN