Software engineering for data science

Foundations of Data Science (INF2–FDS)

Anna Hadjitofi
Semester 2, Week 6
26th Feb 2024
Where does software engineering fit in with data science?
Where does software engineering fit in with data science? **Reproducibility**
Where does software engineering fit in with data science?

**Model design / deployment**

**Meeting latency requirements**
- e.g. a video game company using a model for match-making must have near instantaneous predictions.

**Reduce operational costs**
- If prediction traffic varies throughout time (e.g. food delivery service around dinner), the solution should automatically scale.

**Security**
- Access to models and data should be protected.

**Continuous redeployment & monitoring**
- Automated tests can verify prediction metrics have not drifted out of bounds.

**Transition from training environments**
- Online predictions require real-time data transformations.
"Data scientist" can be a catch-all phrase for a wide range of work.

The analytics - engineering spectrum

Analytics
Statistics, reporting

Data science
Models, pipelines

Software Engineering
Compute, storage
Data scientist job listing on Tesla

- Extensive experience writing software with Python
- Experience with multiple data architecture paradigms (e.g. SQL, NoSQL, Kafka, Spark)
- Experience and interest in frontend development, preferably with the Javascript React framework
- Knowledge of various data communication protocols (e.g. REST API, Websockets)
- Able to work under pressure while collaborating and managing competing demands with tight deadlines
- A passion and curiosity for data and data-driven decision making
- Experience with open source machine learning libraries and frameworks (Tensorflow, Keras, etc)
- Familiarity with continuous integration pipelines (Docker, Jenkins, Kubernetes)
- Drive to introduce a predictive model to a production environment
- Success building and tuning image classification models
- MS in engineering, physics, mathematics, or equivalent.
- 3 - 5 years relevant experience.
- Have high attention to details.
- Be a team player and have the ability to collaborate well across diverse functional groups
- Strong verbal and written communication skills to manage and communicate the health and integrity of the data and systems.
- Experience in high volume manufacturing is a plus

The analytics - engineering spectrum

https://mattsosna.com/DS-transition-1
Data scientist job listing for Yelp

The analytics - engineering spectrum

https://mattsosna.com/DS-transition-1
Upcoming:

Reproducible research / data science workflow

Version control

- Code
- Data
With thanks to...

Books


Websites / blogs:


Barriers to reproducibility
What is reproducibility in research?
Reproducibility is necessary but not sufficient for high quality research.

Caveats: Being reproducible doesn’t mean the answer is right
Barriers to reproducible research (1)

Data not shared / lost / inaccessible format

Missing / buggy code

Code runs but gives different results

Library changes
Putting your code and data online can be revealing and intimidating*

Making an analysis reproducible takes time, particularly at the start of the project. But, you’re helping “future you” and collaborators reuse the work or make changes*

*Illustrations by The Ludic Group LLP from Kirstie Whitaker’s keynote presentation at Scientific Data in 2017. Used under a CC-BY 4.0 license. DOI: 10.6084/m9.figshare.5577340.v1.
Steps towards reproducible data science
Clean code
What is clean code? Quotes from Martin (2009)

Michael Feathers, author of *Working Effectively with Legacy Code*

I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. Clean code always looks like it was written by someone who cares. There is nothing obvious that you can do to make it better. All of those things were thought about by the code’s author, and if you try to imagine improvements, you’re led back to where you are, sitting in appreciation of the code someone left for you—code left by someone who cares deeply about the craft.

Bjarne Stroustrup, inventor of C++ and author of *The C++ Programming Language*

I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

Grady Booch, author of *Object Oriented Analysis and Design with Applications*

Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of crisp abstractions and straightforward lines of control.

Ward Cunningham, inventor of Wiki, inventor of Fit, coinventor of eXtreme Programming. Motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code.

You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the code also makes it look like the language was made for the problem.

**Recommended reading:**
Clean code: simple example

- Variable names should be explanatory and descriptive
- Use standard **formatter** (Python's Black)
- **Follow PEP8 conventions** (case convention, underscore use etc)

```python
def foo(df, c, n):
    v = df[c].value_counts().head(n).index
    return df[df[c].isin(v)]

vs

def filter_by_top_values(clients, column, n_top_values):
    value_counts = clients[column].value_counts()
    top_values = value_counts.head(n_top_values).index
    clients_filtered = clients[clients[column].isin(top_values)]
    return clients_filtered
```
<table>
<thead>
<tr>
<th>Clean code principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface segregation principle</td>
</tr>
<tr>
<td>Keep it simple, stupid</td>
</tr>
<tr>
<td>Magic numbers</td>
</tr>
<tr>
<td>Boy scout rule</td>
</tr>
<tr>
<td>The importance of clean code</td>
</tr>
<tr>
<td>YAGNI</td>
</tr>
<tr>
<td>Error handling and exception management</td>
</tr>
<tr>
<td>Keep your code DRY</td>
</tr>
</tbody>
</table>
Documentation
“Incomplete or confusing documentation” is the most common problem encountered when developing open-source software - GitHub 2017 survey.

https://opensourceurvey.org/2017
Documentation tells us what the project does, how it works, how to use it, issues encountered, how to contribute, details of the datasets.

**LICENSE**: Specifies how/if the project can be used by others.

**README**: Explains why the project is useful and how to get started, required libraries and their versions, and welcomes new community members.

**CONTRIBUTING**: Contributing docs explains what types of contributions are needed and how the process works.

**CODE_OF_CONDUCT**: Sets ground rules for participants’ behaviour and helps to facilitate a friendly, welcoming environment.

**Other documentation**: e.g. tutorials, walkthroughs, or governance policies.
Documentation tells us what the project does, how it works, how to use it, issues encountered, how to contribute, details of the datasets.
Modular code
Modular code

Writing modular code involves breaking down large tasks into smaller, self-contained functions.

- Minimise the duplication of functions, classes, and modules
- Single responsibility principle: a class/function should have only one responsibility
- Modules allow code to be reused by encapsulating them into files that can be imported into other files
Example starter project structure

```
$ tree
.
| LICENSE
| CODE_OF_CONDUCT # Optional, for open source repo
| CONTRIBUTING.md # Optional, for open source repo
| README.md
| requirements.txt # Required libraries and versions for reproducing analysis env
| main.py # Entry script
| data # Data (versions) used for model training/fitting
| data_raw.csv
| data_preprocessed.csv
| models # (Trained) model artefacts or config files
| model_8743b52.pkl
| notebooks
| features.ipynb
| cluster.ipynb
| src # Stores the functions/classes utilised in your pipeline
| __init__.py # Make src a Python module
| kmeans.py
| preprocessing.py
| train.py
| predict.py
| visualise.py
| utility.py
| tests # Unit tests for code maintained with src folder
| test_kmeans.py
| test_preprocessing.py
| test_utility.py
```

This is just a starter guide, keep in mind that every project is unique.
Optimised code
Optimising code in data science: pandas.apply vs iterrows

Know which data structures and methods are faster.
Vectorization is a style of programming that deals with entire arrays instead of individual elements. **Use vector operations** *(numpy)* **over loops** when possible, as it allows the use of optimal and pre-compiled funcs on array objects.

```python
# Element wise operation with Python lists
data = list(range(1000000))
%timeit [value + 5 for value in data]
# Prints:
# 57.2 ms ± 2.12 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

# Convert to numpy array type
import numpy as np
data = np.array(data)
%timeit data + 5
# Prints:
# 1.55 ms ± 73.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```
Optimising code in data science: multiprocessing

Multiprocessing is the ability of a system to support more than one processor at the same time.
Optimising code in data science: `dtypes`

When reading in a csv / json file, pandas infers the column types and defaults to the largest data type (int64, float64, object).

```
airbnb listing data loaded using pandas:

listings.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22552 entries, 0 to 22551
Data columns (total 16 columns):
    id                              22552 non-null int64
    name                            22493 non-null object
    host_id                         22552 non-null int64
    host_name                       22526 non-null object
    neighbourhood_group             22552 non-null object
    neighbourhood                   22552 non-null object
    latitude                        22552 non-null float64
    longitude                       22552 non-null float64
    room_type                       22552 non-null object
    price                           22552 non-null int64
    minimum_nights                  22552 non-null int64
    number_of_reviews               22552 non-null int64
    last_review                     18644 non-null object
    reviews_per_month               18638 non-null float64
    calculated_host_listings_count  22552 non-null int64
availability_365                 22552 non-null int64
dtypes: float64(3), int64(7), object(6)
memory usage: 2.8+ MB
```

availability_365 has only 365 possible values (the number of days each year a listing is available), so it can be downcasted to an int16 without losing info.
airbnb listing data loaded using **pandas**:

```python
listings.info()
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Non-null Count</th>
<th>Dtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>22552</td>
<td>int64</td>
</tr>
<tr>
<td>name</td>
<td>22493</td>
<td>object</td>
</tr>
<tr>
<td>host_id</td>
<td>22552</td>
<td>int64</td>
</tr>
<tr>
<td>host_name</td>
<td>22526</td>
<td>object</td>
</tr>
<tr>
<td>neighbourhood_group</td>
<td>22552</td>
<td>object</td>
</tr>
<tr>
<td>neighbourhood</td>
<td>22552</td>
<td>object</td>
</tr>
<tr>
<td>latitude</td>
<td>22552</td>
<td>float64</td>
</tr>
<tr>
<td>longitude</td>
<td>22552</td>
<td>float64</td>
</tr>
<tr>
<td>room_type</td>
<td>22552</td>
<td>object</td>
</tr>
<tr>
<td>price</td>
<td>22552</td>
<td>int64</td>
</tr>
<tr>
<td>minimum_nights</td>
<td>22552</td>
<td>int64</td>
</tr>
<tr>
<td>number_of_reviews</td>
<td>22552</td>
<td>int64</td>
</tr>
<tr>
<td>last_review</td>
<td>18644</td>
<td>object</td>
</tr>
<tr>
<td>reviews_per_month</td>
<td>18638</td>
<td>float64</td>
</tr>
<tr>
<td>calculated_host_listings_count</td>
<td>22552</td>
<td>int64</td>
</tr>
</tbody>
</table>

Use `downcast` arg of `pd.to_numeric` to downcast the data to the smallest dtype possible.

**Optimising code in data science: dtypes**
Logging
Logging

Monitor the flow that our program is going through.

Logging vs print statements:

- Logging allows you to add context (time, location, level)
- Send logs to different places & formats
- Control behaviour via configs
Logging (Python)

Logging to standard output stream:

```python
import logging

logging.basicConfig(
    format="%(name)s - %(asctime)s - %(levelname)s: %(message)s",
    level=logging.INFO,
)

logging.debug("This will only print if level is set to debug")
logging.info("Prints in debug + info levels")
logging.warning("Prints in debug + info + warning levels")
logging.error("Prints in debug + info + warning + error levels")
```

Use appropriate logging level:
Logging (Python)

Logging to a file:

```python
import logging

logging.basicConfig(
    filename='experiment.log',
    encoding='utf-8',
    format='%(asctime)s - %(levelname)s: %(message)s',
    level=logging.INFO,
)

logging.debug("This will only print if level is set to debug")
logging.info("Prints in debug + info levels")
logging.warning("Prints in debug + info + warning levels")
logging.error("Prints in debug + info + warning + error levels")
```

Use appropriate logging level:

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET
Testing
The $125 million satellite was supposed to be the first weather observer on another world. But as it approached the red planet to slip into a stable orbit Sept. 23, the orbiter vanished. Scientists realized quickly it was gone for good. “It was pretty clear that morning, within half-an-hour, that the spacecraft had more or less hit the top of the atmosphere and burned up,” recalled NASA engineer Richard Cook, who was project manager for Mars exploration projects at the time.

A NASA review board found that the problem was in the software controlling the orbiter’s thrusters. The software calculated the force the thrusters needed to exert in pounds of force. A separate piece of software took in the data assuming it was in the metric unit: newtons.

Testing

- **Unit testing:** aims to check if a part of code operates in the intended way.
- **Integration testing:** verifies how different components interact and function together smoothly as a whole.
- **Data testing:** validates the quality, integrity, and consistency of data used in models and analyses.
- **Model testing:** evaluates the performance and generalisability of models on unseen (or incoming) data.
Testing

```python
import re
import pandas as pd

df = pd.DataFrame({"text": ["5 euro", "7 euro"]})

def extract_money(text):
    """Extract monetary value from string by looking for
    a pattern of a digit, followed by 'euro'.
    e.g. 5 euro --> 5
    Args:
        text (str): Text containing monetary value
    Returns:
        float: The extracted value
    """
    extracted_money = re.search(r"\d euro", text).group(1)
    return float(extracted_money)

df["money"] = df["text"].apply(lambda x: extract_money(x))
```

New data incoming:

```python
df = pd.DataFrame({
    "text": ["5 euro", "7 euro", ""],
    "row_number": [1, 2, 3],
})
```

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'NoneType' object has no attribute 'group'
Testing

Fixed code:

```python
import re
import pandas as pd

df = pd.DataFrame({'text': ['5 euro', '7 euro', '']})

def extract_money(text):
    """Extract monetary value from string by looking for a pattern of a digit, followed by 'euro'.
    e.g. 5 euro --> 5
    Args:
        text (str): Text containing monetary value
    Returns:
        float: The extracted value
    """
    if text:
        extracted_money = re.search(r'\d+ euro', text).group(1)
        return float(extracted_money)
    else:
        return None

df["money"] = df["text"].apply(lambda x: extract_money(x))
```

Corresponding test case:

```python
from src.money_manager import extract_money
import pytest

def test_empty_string():
    empty_string = ""
    extracted_money = extract_money(empty_string)
    expected_output = None
    assert extracted_money == expected_output

# Run using pytest -v
```

Note that any numbers with decimal points would still fail this test!
Refactoring
When to refactor (data science / ML projects)?

**Model drift**
If performance drops, it may call for a retraining or refactoring to better reflect any changes to the environment.

**New maintainer**
When taking on a project someone else built or vice versa, evaluating whether a refactor would be of value (and doing one) can be helpful for a handover.

**Change of source data**
Changes in features, volume of data or how it’s measured.

**Scaling**
Shift in requirements of the pipeline (users, new data).

**Moving from R&D to production**
Ensure model integrates with pipeline, & improve performance for scale.
Refactoring

Improve the design, structure, and implementation of the code while preserving its functionality.

In general, prerequisites of refactoring:
- Doesn’t change external behavior
- Changes code’s internal structure
- Is done after the code fulfills the requirements

Different methods for refactoring:
red-green refactoring, extract method, simplifying methods, composing method, and abstraction.

This illustration is created by Scriberia with The Turing Way community. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807
Version control
Code management
Source code version control

Tracks & manages changes in a code base.

Insights from exploratory analysis

Scalable models that drive development of services

Artefacts e.g. file dependencies, software versions, datasets, models, metrics and parameters

This illustration is created by Scriberia with The Turing Way community. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807
Git

Linus Torvald’s first commit developing git

For info on the concepts behind Git, see tutorials at:
https://github.com/infpals and
https://homepages.inf.ed.ac.uk/s1334591
Jupyter notebooks are written in **JSON** and generate files that may contain metadata, source code, formatted text, and rich media.

**Diff example of notebooks:**

```
$ diff a.ipynb b.ipynb
76,77d75
< "plt.rc('axes', grid=False)"
< "plt.rc('axes', facecolor='white')"
9c88
< "image/png": "iV80Rw9KgoAAAAANSuHEugAABLkAAAMQCAAYAADLj7dIAAAAAHNCQVICAgIFAhk
AAAAAwSFl2\n\nAAWQQAFIUBS1kI8AIAAJBREFu3j3svXexYFdS7b12h8maPNJII2LGOaCakEBCFgqzIXxBAp
\nN1waoDyZxG8zXzMMzKU2f4Mx1s6PwAwxnmBjg4YH12BFQMa2b9fYAFKikjXKwRt3lI3tS3TzXuX+8Vv2m\n\n9z/o9zzyunpvrq1L6nqyqtqPprbRNFEQghhBBCCGEEEIIISkhlwMghBBCCGEEEIIISQ\n\nBC1pQ5CkCEEEIIYYQQQggh3k3OrxBCGCCGEEEIIYR4D9UuQggghBBCCGGEEI3\n\nEIIIYQQQggh3k3OrxBCGCCGEEEIIYR4D9UuQggghBBCCGGEEI3\n\nEEIIIIYYQQQggh3k3OrxBCGCCGEEEIIYR4D9UuQggghBBCCGGEEI3\n\n```
Notebooks vs programs

Use tooling for diffing & merging Jupyter notebooks, e.g. Git integrations in VSCode or nbdime.

Also, for large notebooks with many image outputs:

- Clear output manually
- Convert to HTML
- Convert to Python (script)
Data management
Data version control

Version control systems deal well with small text files (kb instead of mb, and definitely not gb (Wilson et al., 2017)).

Recommendations:

- **Save and backup the raw data**, protect with permissions and document how it was obtained (e.g. exact query, date of retrieval, version of database)

- **Save and share a clean version of the data** in open data format (`csv`, `json`, `yaml`, `xml`) with meaningful variable and file names, as well as metadata
Data version control

Tidy dataset:

- Every column is a variable
- Every row is an observation
- Every cell is a single value
- Ideally, unique ID for each observation

Covered in Section 2.2 of FDS lecture notes

Share data using open access research data repos: e.g. Zenodo, figshare, Mendeley Data
Happy to take any questions.

Feel free to get in touch with future questions or any feedback on the session: a.hadjitofin@ed.ac.uk

https://forms.office.com/e/mWK1u5cXgT
Optimising code in data science: **vectorise your functions**

**Data types**
Unlike Python lists, `numpy` allows arrays to only have a single data type and stores the data internally in a contiguous block of memory.

**Broadcasting**
A feature of `numpy` that enables mathematical operations to be carried out between arrays of different sizes (allows vectorising array operations so that looping occurs in C instead of Python).

**Vectorisation cannot be applied:**
- Loop dependency
- Indirect memory access
- Code branching
Refactoring (methods)

**Red-green** refactoring. “Test first approach”. Review intended development and write tests (red), implement code (green) and then identify weak points and refactor.

**Abstraction.** Remove repetition and redundancy from your code, e.g. creating interfaces, setting up new classes, hierarchy, class inheritances, etc.

**Composing** method. Long methods make code hard to understand and sometimes change. Transfer a code fragment from its original method into a newly established one (**extraction**).

**Simplifying** methods. Addresses complicated logic. Consolidate multiple conditionals that lead to the same result or action to a single expression (**conditional expressions refactoring**). Adding / removing parameters, or replacing parameters with explicit method and call (**methods calls refactoring**).