Software engineering for data science

Foundations of Data Science (INF2-FDS)

Anna Hadjitofi

Semester 2, Week 6

26th Feb 2024

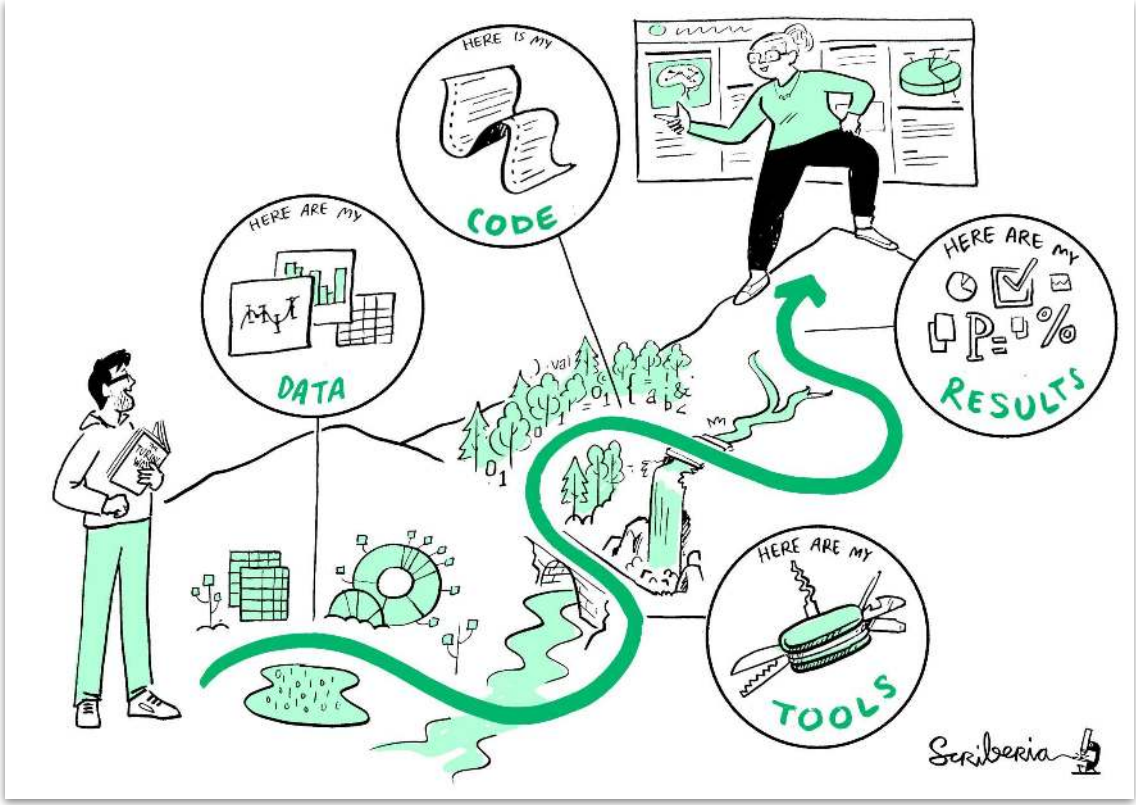
THE UNIVERSITY of EDINBURGH
informatics

Where does software engineering fit in with data science?

Modularity

Readability

Robust (testing, errors, logging)



Efficient & maintainable code

Packaging / sharing your code

Where does software engineering fit in with data science? **Reproducibility**

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

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

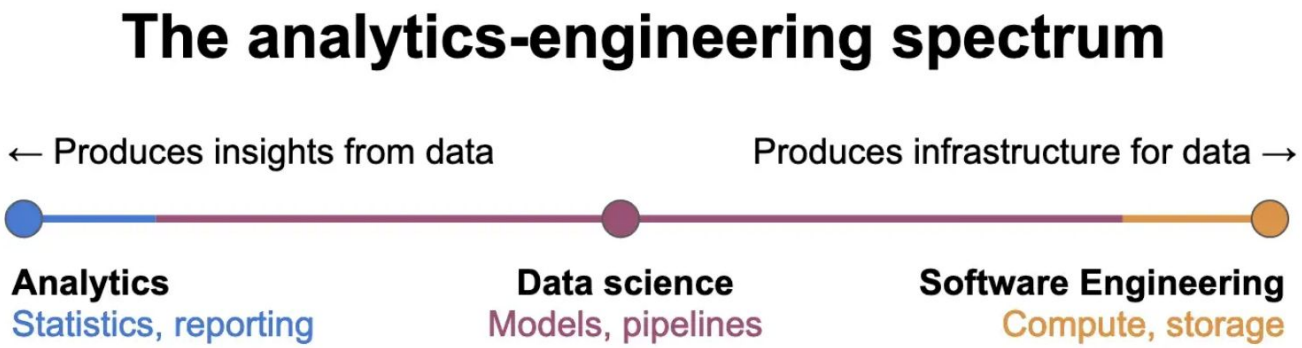
Security
Access to models and data should protected.

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.

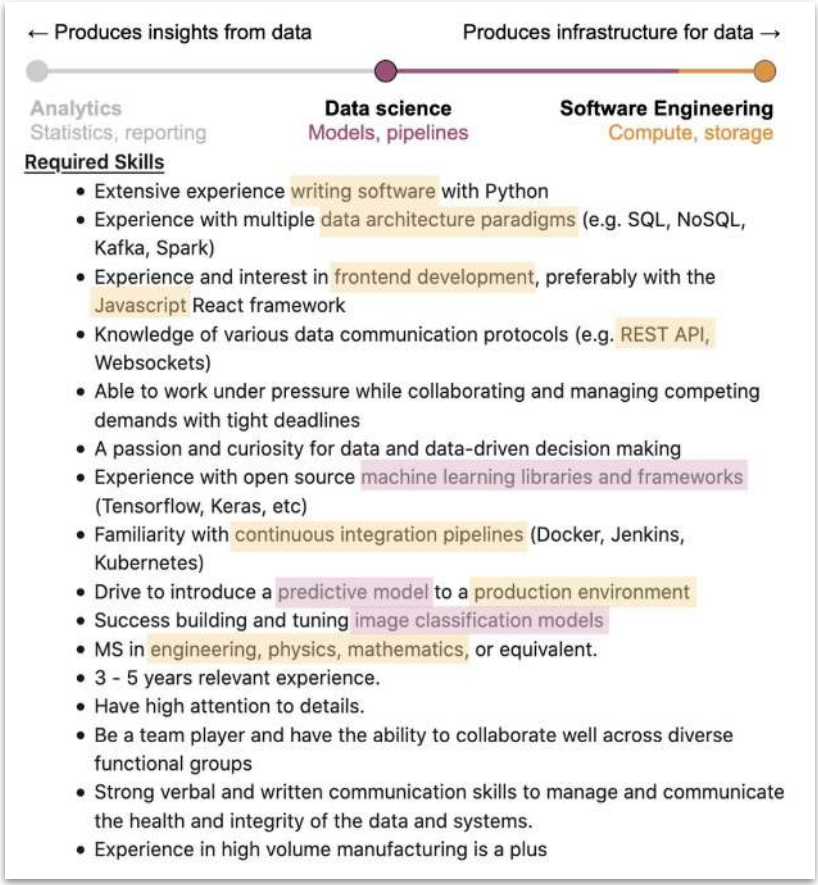
Where does software engineering fit in with data science? **Model design / deployment**

“Data scientist” can be a catch-all phrase for a wide range of work.



By Matt Sosna

Data scientist job listing on Tesla



← Produces insights from data — Produces infrastructure for data →

Analytics
Statistics, reporting

Data science
Models, pipelines

Software Engineering
Compute, storage

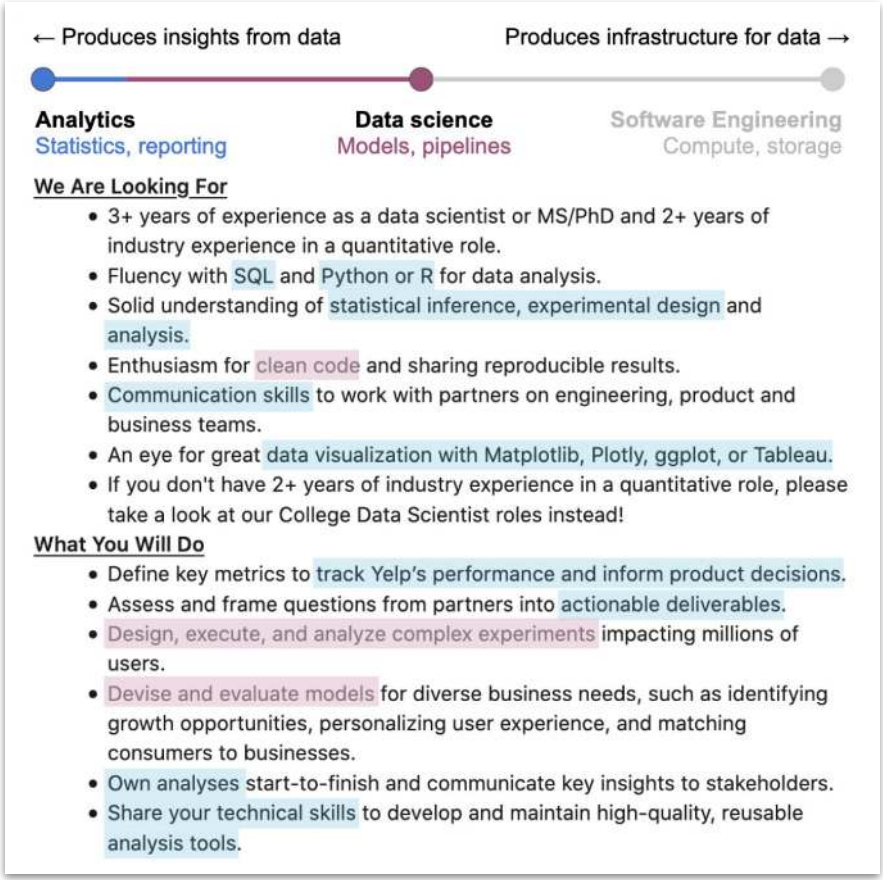
Required Skills

- 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



← Produces insights from data — Produces infrastructure for data →

Analytics Statistics, reporting | **Data science** Models, pipelines | **Software Engineering** Compute, storage

We Are Looking For

- 3+ years of experience as a data scientist or MS/PhD and 2+ years of industry experience in a quantitative role.
- Fluency with SQL and Python or R for data analysis.
- Solid understanding of statistical inference, experimental design and analysis.
- Enthusiasm for clean code and sharing reproducible results.
- Communication skills to work with partners on engineering, product and business teams.
- An eye for great data visualization with Matplotlib, Plotly, ggplot, or Tableau.
- If you don't have 2+ years of industry experience in a quantitative role, please take a look at our College Data Scientist roles instead!

What You Will Do

- Define key metrics to track Yelp's performance and inform product decisions.
- Assess and frame questions from partners into actionable deliverables.
- Design, execute, and analyze complex experiments impacting millions of users.
- Devise and evaluate models for diverse business needs, such as identifying growth opportunities, personalizing user experience, and matching consumers to businesses.
- Own analyses start-to-finish and communicate key insights to stakeholders.
- Share your technical skills to develop and maintain high-quality, reusable analysis tools.

The analytics - engineering spectrum

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

Upcoming:

Reproducible research / data science workflow

Version control

- Code
- Data

With thanks to...

Books

The Turing Way Community. (2021, November 10). The Turing Way: A handbook for reproducible, ethical and collaborative research. Zenodo. http://doi.org/10.5281/zenodo.3233853.

Martin, R. C. (2009). Clean code: a handbook of agile software craftsmanship. Pearson Education.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLoS computational biology, 13*(6), e1005510.

Websites / blogs:

W.D., “Scikit-learn's Defaults are Wrong”. 2019. https://ryxcommar.com/2019/08/30/scikit-learns-defaults-are-wrong

“Open Source Survey”. 2017. https://opensourcesurvey.org/2017

Sosna, M., “How to enter data science”. 2020. https://mattsosna.com/DS-transition-1

Van der Gugten, R., “Advanced Pandas: Optimize speed and memory”. 2019. https://medium.com/bigdatarepublic/advanced-pandas-optimize-speed-and-memory-a654b53be6c2

Kirmer, S., “Refactoring Machine Learning Projects”. 2021. https://towardsdatascience.com/refactoring-machine-learning-projects-f566607a7b6f

Barriers to reproducibility

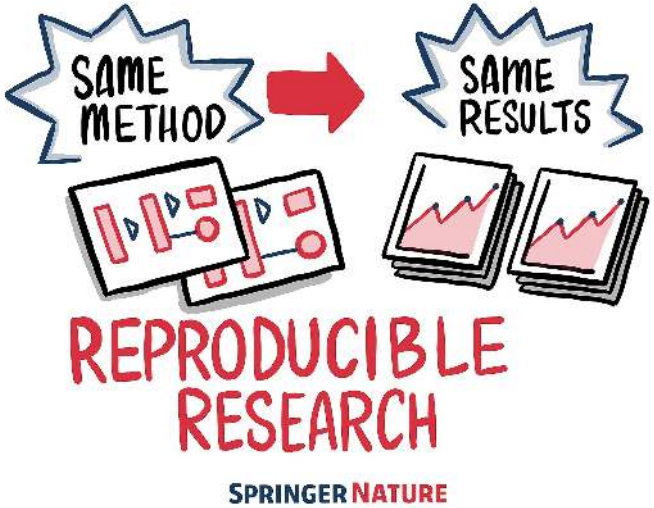


Illustration by The Ludic Group LLP from Kirstie Whitaker's keynote presentation at Scientific Data in 2017. Used under a CC-BY 4.0 license. DOI:10.6084/m9.figshare.5577340.v1.

		Data: Same	Data: Different
Analysis	Same	Reproducible	Replicable
Analysis	Different	Robust	Generalisable

The Turing Way Community. (2021, November 10). The Turing Way: A handbook for reproducible, ethical and collaborative research. Zenodo. http://doi.org/10.5281/zenodo.3233853

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

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*

Barriers to reproducible research (2)

*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

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.

What is clean code? Quotes from Martin (2009)

Recommended reading: Martin, R. C. (2009). Clean code: a handbook of agile software craftsmanship. Pearson Education.

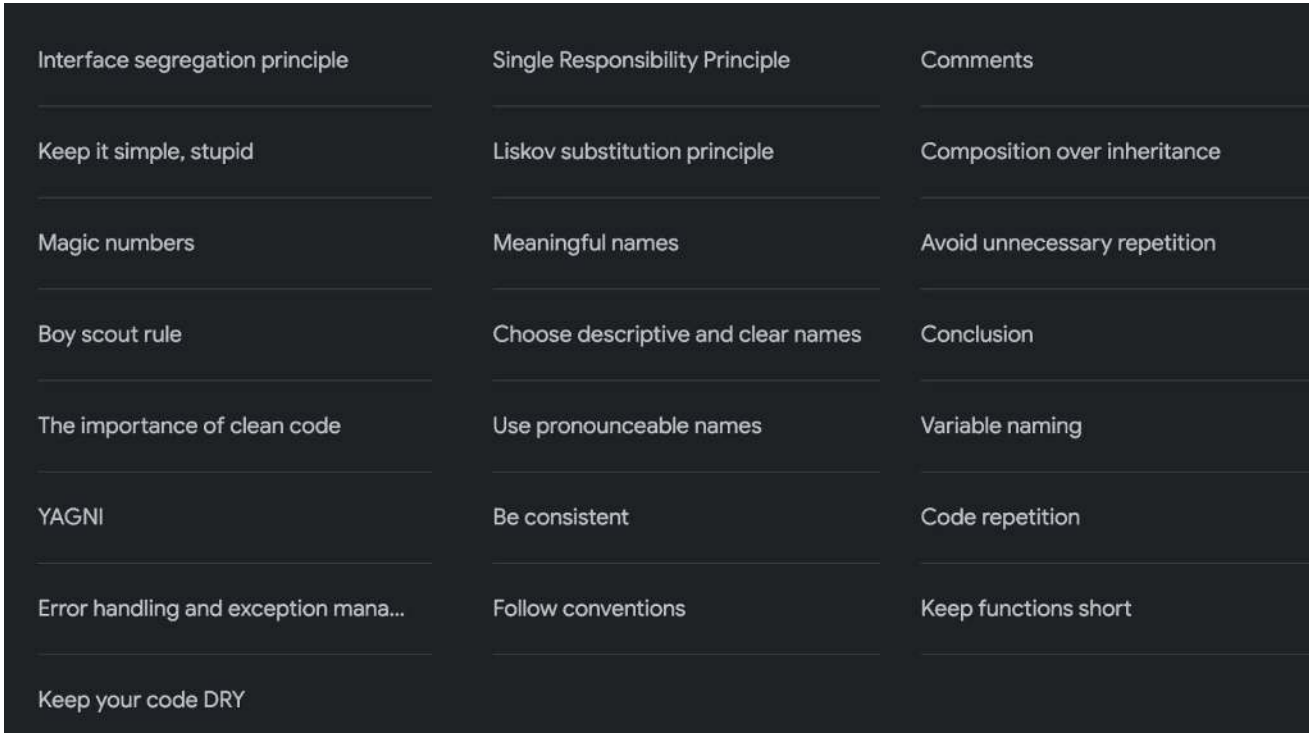
```python
def foo(df, c, n):
    v = df[c].value_counts().head(n).index
    return df[df[c].isin(v)]
```

vs

```python
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
```

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

Clean code: simple example

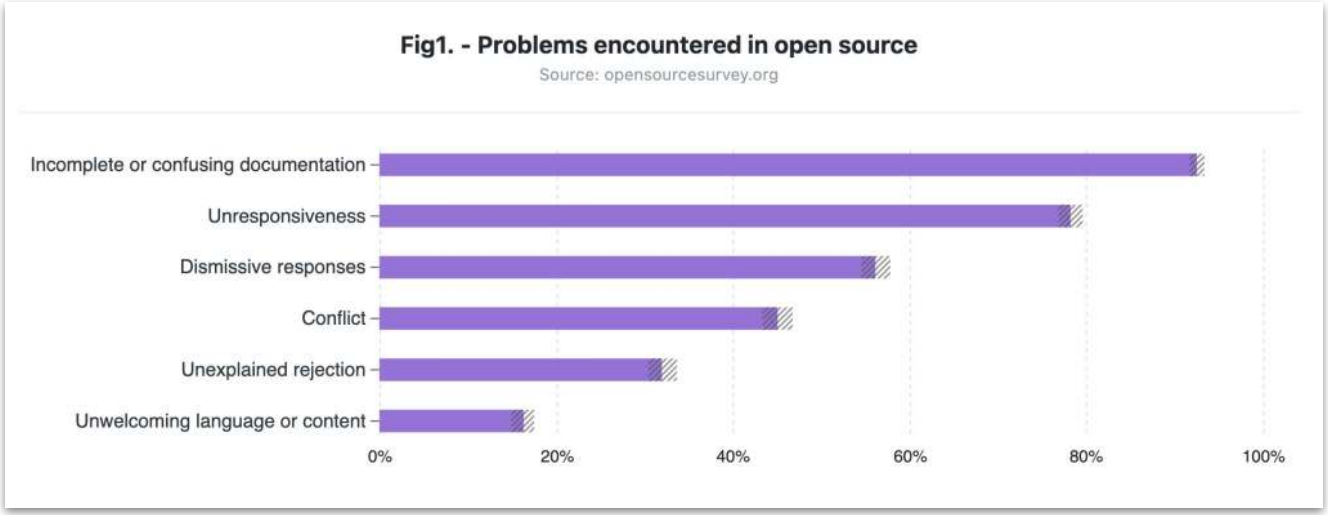


Clean code principles

Documentation

Documentation

“Incomplete or confusing documentation” is the most common problem encountered when developing open-source software - GitHub 2017 survey.



https://opensourcesurvey.org/2017

Documentation

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

Documentation tells us what the project does, how it works, how to use it, issues encountered, how to contribute, details of the datasets.

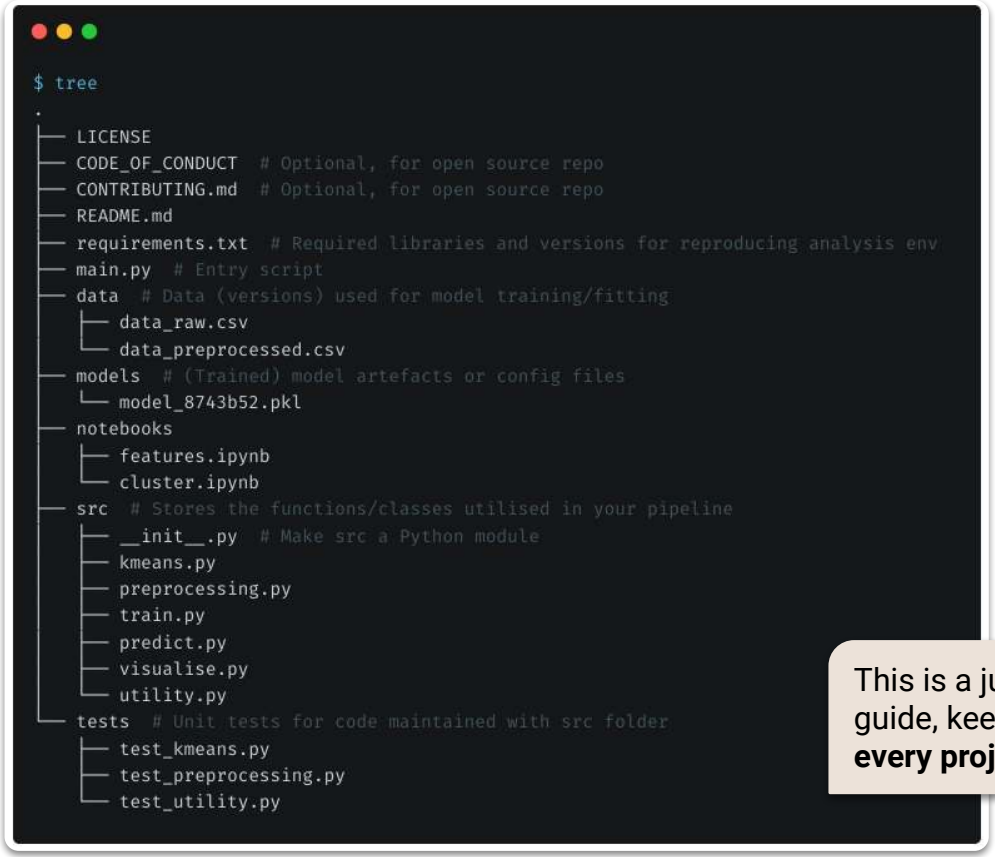
```
1577836804,230,230,229,1420,1152,1600,32099,25194,36533,39,31,44
1577837106,229,230,229,1391,1144,1612,31736,25095,36790,38,31,44
1577837404,229,230,229,1443,1149,1631,32153,25503,36212,39,31,45
1577837704,229,230,229,1446,1142,1641,32290,24880,36033,39,31,45
1577838004,230,230,229,1416,1176,1638,32536,25760,37207,39,32,45
1577838304,229,230,229,1431,1154,1624,31684,25918,36888,39,31,44
1577838603,229,229,229,1398,1162,1652,32399,25318,37848,38,32,45
1577838905,229,230,229,1446,1163,1633,32165,25621,37040,39,32,45
1577839204,229,230,230,1438,1109,1629,32561,25796,36723,39,30,44
1577839504,229,229,229,1465,1169,1604,31812,25417,36719,40,32,44
1577839804,229,229,230,1423,1114,1612,32291,25185,36687,39,30,44
1577840104,229,230,229,1435,1172,1637,32502,25012,36815,39,32,45
1577840404,229,230,229,1410,1172,1590,32488,25131,37004,38,32,43
```

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



This is a just a starter guide, keep in mind that **every project is unique.**

Example starter project structure

Optimised code

```python
import time
import math

import numpy as np
import pandas as pd

def myfunc(n):
    count = 0
    for i in range(1, (int)(math.sqrt(n)) + 1):
        if n % i == 0:
            if n / i == i:
                count = count + 1
            else:
                count = count + 2
    return count

random_ints = np.random.default_rng().integers(low=0, high=10, size=100000).tolist()
data = pd.DataFrame({"randInts": random_ints})
```

```python
# Using iterrows
start_time = time.time()
for i, row in data.iterrows():
    myfunc(row["randInts"])
print(time.time() - startTime) # → 3.7 seconds
```

```python
# Using apply
start_time = time.time()
_ = data["randInts"].apply(lambda x: myfunc(x))
print(time.time() - startTime) # → 0.08 seconds
```

Know **which data structures and methods are faster**

Optimising code in data science: `pandas.apply vs iterrows`

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.

```
# 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: **vectorise your functions**

```python
import math
import multiprocessing as mp
import time

import numpy as np
import pandas as pd

def myfunc(n):
    """Apply some function to a given number."""
    count = 0
    for i in range(1, (int)(math.sqrt(n)) + 1):
        if n % i == 0:
            if n / i == i:
                count = count + 1
            else:
                count = count + 2
    return count

if __name__ == "__main__":
    random_ints = np.random.default_rng().integers(low=0, high=10, size=5000000).tolist()
    data = pd.DataFrame({"randInts": random_ints})
    startTime = time.time()
    answer = data["randInts"].apply(myfunc)
    print(time.time() - startTime)  # → 3.3 seconds
```

```python
import math
import multiprocessing as mp
import time

import numpy as np

def myfunc(n):
    """Apply some function to a given number."""
    count = 0
    for i in range(1, (int)(math.sqrt(n)) + 1):
        if n % i == 0:
            if n / i == i:
                count = count + 1
            else:
                count = count + 2
    return count

if __name__ == "__main__":
    random_ints = np.random.default_rng().integers(low=0, high=10, size=5000000).tolist()
    startTime = time.time()
    with mp.Pool(processes=mp.cpu_count() - 1) as pool:
        answer = pool.map(myfunc, random_ints)
    print(time.time() - startTime)  # → 1.4 seconds
```

Multiprocessing is the ability of a system to **support more than one processor at the same time**.

Optimising code in data science: **multiprocessing**

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

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
```

```
pd.to_numeric(listings["availabiliy_365"], downcast="integer")
```

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

Logging

Logging

Monitor the flow that our program is goes 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")
```

```
>root - 2023-02-11 23:00:31,626 - INFO: Prints in debug + info levels
>root - 2023-02-11 23:00:31,626 - WARNING: Prints in debug + info + warning levels
>root - 2023-02-11 23:00:31,626 - ERROR: Prints in debug + info + warning + error levels
```

Use appropriate logging level:

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

Logging (Python)

Logging to a file:

```python
import logging

logging.basicConfig(
    filename="experiment.log",
    encoding="utf-8",
    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:

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

Testing

Testing

The New York Times

FATAL RADIATION DOSE IN THERAPY ATTRIBUTED TO COMPUTER MISTAKE

Share full article

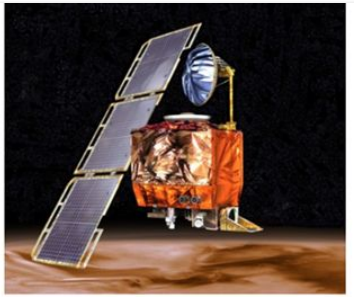
AP

June 21, 1986

https://www.nytimes.com/1986/06/21/us/fatal-radiation-dose-in-therapy-attributed-to-computer-mistake.html

LISA GROSSMAN 11.10.10 7:00 AM

NOV. 10, 1999: METRIC MATH MISTAKE MUFFED MARS METEOROLOGY MISSION



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*.**

https://www.wired.com/2010/11/1110mars-climate-observer-report/

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 in-coming) 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("(\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

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

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

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

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.

When to refactor (data science / ML projects)?

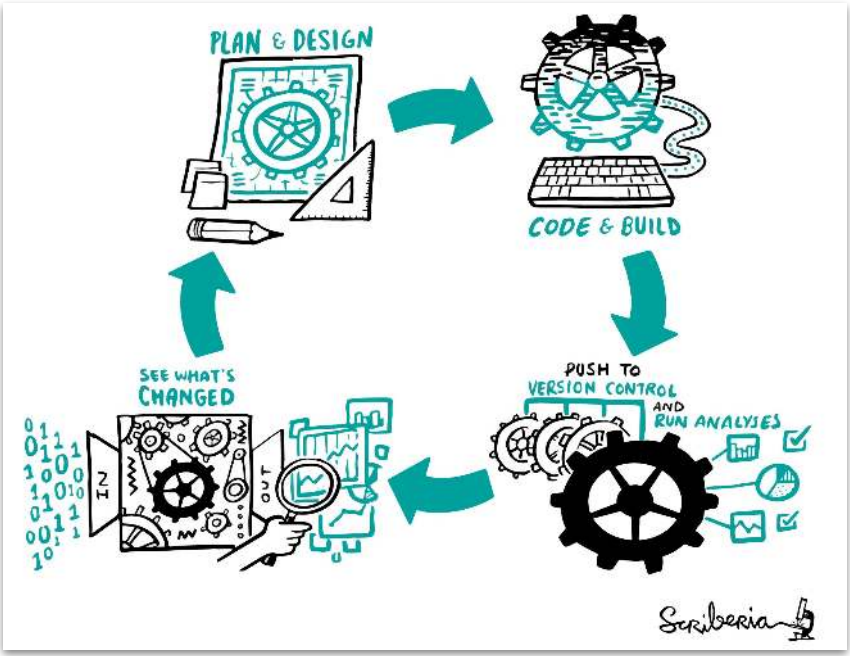
Refactoring

In general, prerequisites of refactoring:

- Doesn't change external behavior
- Changes code's internal structure
- Is done after the code fulfills the requirements

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

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



Version control

Code management

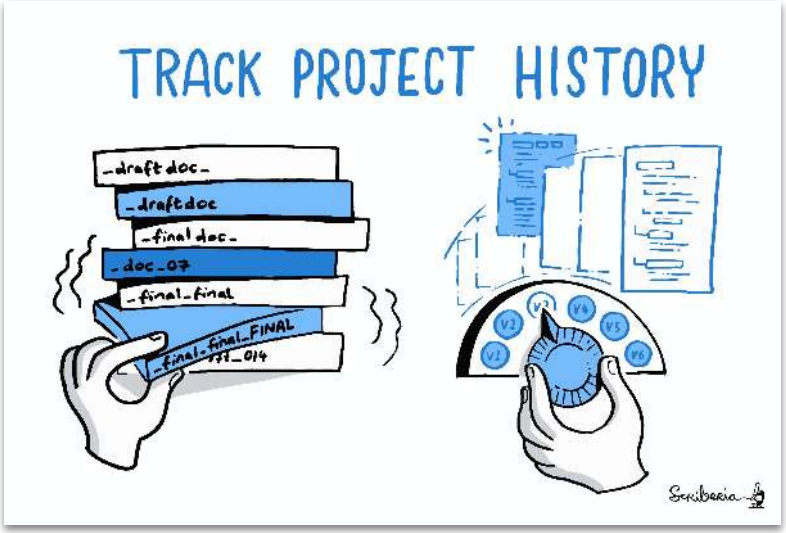
Source code version control

Tracks & manages changes in a code base.

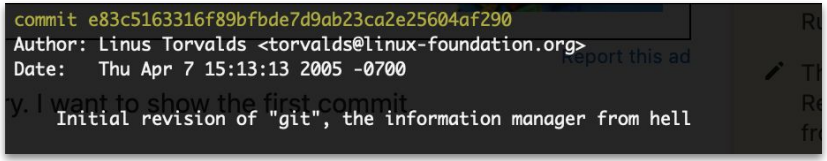
Insights from exploratory analysis

Scalable models that drive development of services

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



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

Notebooks vs programs

Primitive line-based diff and merge tools work best on plain text.

Jupyter notebooks are written in `JSON` and generate files that may contain metadata, source code, formatted text, and rich media.

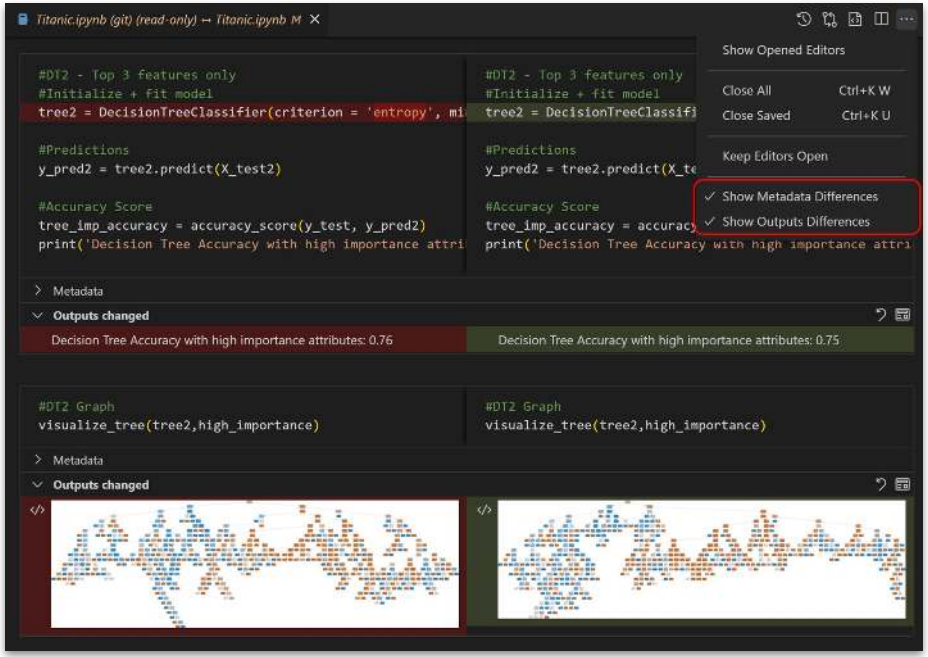
Diff example of notebooks:

```
$ diff a.ipynb b.ipynb
76,77d75
<     "plt.rc('axes', grid=False)\n",
<     "plt.rc('axes', facecolor='white')\n",
90c88
<       "image/png": "iVBORw0KGgoAAAANSUhEUgAABLkAAAMQCAYAAADLj7dlAAAABHNCSVQICAgIfAhki
AAAAAlwSFlz\nAAAWJQAAFiUBSVIk8AAAIABJREFUeJzsvXeYZFd57b12h0maPNJII2lGOaCAkEBCFgozIxkBAp
lY\n1waDyDZg8MX+zMU2F4Mx1x8PwWAwxmBjg4yNi2BfQMa20iiAQFkIjXKWRtJIE3tSz3TXuX+8vV2n\nqyucv
N+9z/o9zzynprvq1D6nqqtqr1prbRNFEQghhBBCCCGEEEII8Zkh1wMghBBCCCGEEEIIISQv\nFLkIIYQQQgghhB
BCiPdQ5CKEEEIIIYQQQggh3kORixBCCCGEEEIIIYR4D0UuQgghhBBCCCGEEOI9\nFLkIIYQQQgghhBBCiPdQ5CK
EEEIIIYQQQggh3kORixBCCCGEEEIIIYR4D0UuQgghhBBCCCGEEOI9\nFLkIIYQQQgghhBBCiPdQ5CKEEEIIIYQQ
Qggh3kORixBCCCGEEEIIIYR4D0UuQgghhBBCCCGEEOI9\nFLkIIYQQQjzEGHOJMaZljPmo67EkZWq8D7keByGEE
ELChCIXIYQQQirDGPOmKaFj3BhzkMNx/H/G\nmG3GmP/pagwFEbkeQJUYY75gjNlijHmD67EQQghRB8UuQghhB
BSJe+DCDMjAH7L4TjeAmA+gLc5\nHEMRGNcDqJi3AVgI4DddDD4QQQggh+qDIRQghhJBKMMacCuBMAFsg4sy7jTH
DjobzZwBuBvBxR/dP\nsvERADcC+LTrgRBCCCFEHxS5CCGEEFIVH4C4uP4SIlQcQcBOD1LgYSRVEziqIXR1H0fRf3
T7IRRdFf\nRlH0K1EUXe96LIQQQgjRB0UuQgghhJSOMWYpgP8BoAXg7wH8HHcTN9Tsux0UIIYQQQsKBIhchhBBC\
```

Notebooks vs programs

Rmarkdown files include code and prose (results produced by code are processed / typeset to produce an additional `.pdf` or `.html` file)

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:

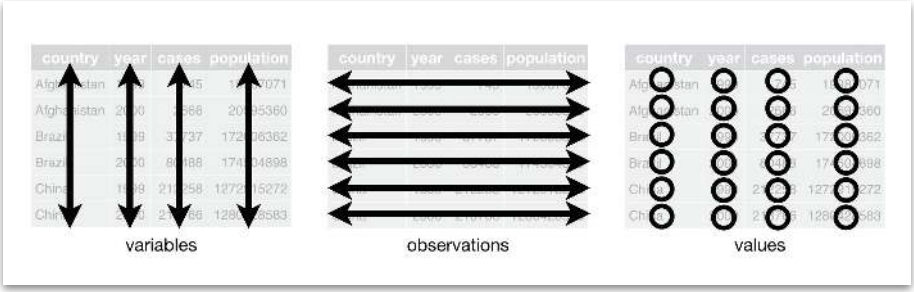
- **Save and backup the raw data**, protect with permissions and document how it was obtained (e.g. exact query, date of retrieval, version of database)
- **Save and share a clean version of the data** in open data format (`csv`, `json`, `yaml`, `xml`) with meaningful variable and file names, as well as metadata

Data version control

Share data using open access research data repos: e.g. Zenodo, figshare, Mendeley Data

Tidy dataset:

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



See *Tidy data in R*

Covered in Section 2.2 of FDS lecture notes

FOUNDATIONS OF DATA SCIENCE

Happy to take any questions.

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

https://forms.office.com/e/mWK1u5cXgT

THE UNIVERSITY of EDINBURGH
informatics

Extra slides

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**).