IMPLEMENT
CONSULTING GROUP_

WHAT IS DATA SCIENCE?

What is data science?

The big data revolution is now at least a decade old, and every company and public institution has more or less fully developed data warehouses. But what are they being used for? What value are they creating? We have gotten better and more detailed reporting on the daily operations, more management reporting and more dashboards. All of this is good and valuable, but we can do so much more. This is what data science is all about.

Data science promises to bring operational solutions and even deeper insights into organisations. As such, the promise of data science is closer to the original promise of mechanical automation. Data science produces solutions. Solutions that go into productions and directly affect an operational part of the organisation, supporting, improving or maybe automating a work process. Sometimes even enabling new work processes. This is not the only feature of data science, but it is probably the core promise, and why it is being compared to the fourth industrial revolution (Forum 2016).

An example could be a doctor. A patient comes through the door, and a data science solution not only presents a dashboard based on the patient's journal but also a set of predicted points of attention for the doctor based on the patient's history, the latest research and the current disease landscape in the world. The patient gets a scan of his lungs which are causing trouble, and as the scanner delivers the image, it immediately suggests that a region of the image be examined for potential cancerous growth and immediately highlights the area. The machine is trained on millions of images worldwide, and the doctor is immediately more alert. After the patient has left, the doctor dictates a note which is automatically saved as text. An algorithm detects that a scan of the lungs is mentioned and automatically adds a procedure code to the patient's medical history.

DATA SCIENCE IS AN INTERDISCIPLINARY FIELD COLLECTING CLASSICAL STATISTICS, DATA ANALYSIS AND MACHINE LEARNING METHODS IN AN ATTEMPT TO UNDERSTAND AND ANALYSE REAL-WORLD PHENOMENA VIA DATA.

It is also noted that cancer is suspected, and a flag of attention is raised on the patient for further attention.

In the example above, data science was first used to give the user an overview based on an otherwise incomprehensibly large information base. Subsequently, decision support was provided as the area was highlighted. Finally, there was a complete automation of the medical record keeping of the work. We are hardly there yet. But I have no doubt that we are on the way. This guide will give you an idea of how data science works in organisations today, and how you can get started in your own organisation.

Who is this guide for?
This guide is first and foremost for people who want to increase their organisation's data science skills. Either by becoming better data scientist or software developers themselves or by leading a data science department, small or large. This guide provides deep insight into a complete data science workflow – not just from data to model but from identifying the right problem to setting up the model to realising value and ensuring maintenance.

Why this guide?
Though many organisations want to get started with artificial intelligence and data science, and some even try, they systematically encounter some challenges. These are the challenges we would like to address in this guide.

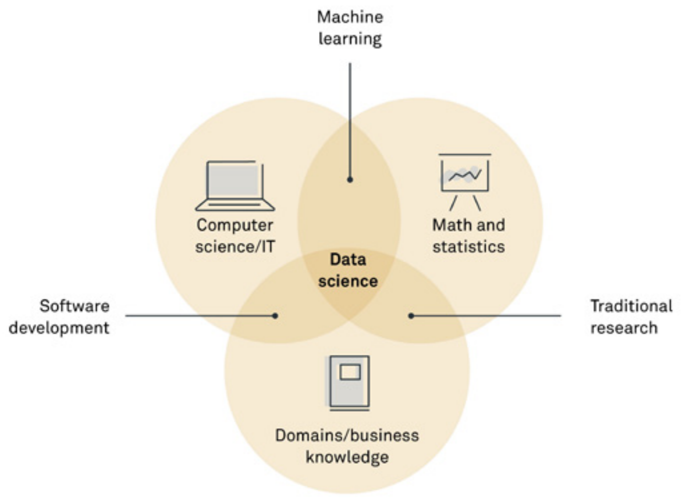
Inspiration

In Implement, we experience a great demand for data science competencies. This is driven by the rapid growth of data in most modern organisations, which naturally raises the question: How can we leverage all this data?

The next thing that usually happens is that an employee is set to work. Either to identify a use case or is given a concrete one by management.

The challenge is that a skilled data scientist has three skills which is an extremely rare combination: (1) a strong understanding of software development, (2) a strong understanding of statistics and machine learning and (3) a strong understanding of the business or IT domain they work in.

Finding the right employee (or candidate) with this set of skills can be very tricky, and developing a department with it introduces its own challenges.



If one of these elements is missing, we see some common challenges arise:

- Lacking **software developer skills**, the data scientist risks becoming an expert who cannot deliver solutions that are actually implemented and bring about change. The solutions remain too technical or are often just never finished.
- Lacking **statistics and machine learning skills**, the data scientist will often find it difficult to provide the adequate quality of solutions in demand. They will probably finish and solve the problem, but they are prevented from tackling the new and difficult problems which they potentially could if they had a greater understanding of machine learning.

- Lacking **business knowledge**, they risk developing great solutions that are not applicable in business and where the benefits of development are never realised.

Not surprisingly, finishing projects, achieving the expected quality and realising the desired impact are some of the most widespread problems in new data science projects.

Content of this guide

In this guide, you will be taken on a journey from the earliest start of a data science project with use case identification to advising on governance of models. The guide consists of seven chapters with the following headings:

- **Chapter 1. Use case identification:** How do you ensure that you are working on a valuable idea?
- **Chapter 2. Data exploration:** Get an overview of your data from patterns and content to creation and management.
- **Chapter 3. Definition of purpose:** What should your model optimise for? What exactly is the problem we are trying to solve? And how do you make sure that your model actually does what you expect?
- **Chapter 4. Machine learning:** What techniques do we use to solve the problems? How do you hit the right amount of complexity and ensure that your model works in real life?
- **Chapter 5. Ethics:** Data science touches on many human aspects from personal data to automation. Chapter 5 gives an introduction to the main considerations as well as some concrete advice on how to proceed in a human-centric way.
- **Chapter 6. Data science governance:** When models are designed to create value, they must be managed and maintained. They may even need to be updated or at least not burden the IT landscape around them. That is why we have governance.
- **Chapter 7. Governance tools:** When we implement governance around models, we look for some helpful tools. Chapter 7 provides an insight into the five most important tools from version control to model monitoring.

Chapter 1. Use case identification

How do we ensure that our data science solutions actually make an impact? This is often easier said than done, and data science has some special pitfalls to keep in mind. Part of the challenge is probably that the professional overlap between people who are familiar with the latest opportunities in machine learning and AI is that they are often quite young and have not necessarily studied innovation processes or business development. There is an (understandable) fascination with the technology, which I myself can sometimes fall for too. This challenge means that many data science projects are getting started with the wrong problem which either does not solve anything really valuable or proves impossible to solve.

What is a (data science) use case?

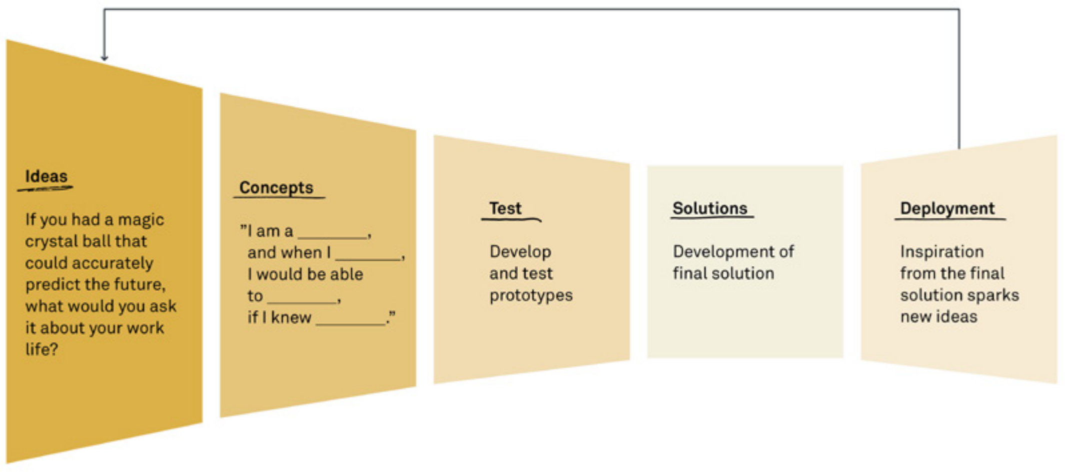
A use case is two things: a goal and a path to it. Alternatively: a problem and a solution. An example could be translating texts from French to English, and the solution could be a machine translation system such as Google Translate.

Use case identification is about first identifying interesting and valuable goals and problems and then identifying well-designed solutions for them.

A place to start but not to stop

If someone out there reads this and feels that I overlook a lot of possibilities (e.g. unsupervised learning or reinforcement learning), the answer is yes. The tools presented here do not make up an exhaustive list for finding every data science use case in an organisation, but it is a way to get started.

Use case identification


Ideas
If you had a magic crystal ball that could accurately predict the future, what would you ask it about your work life?
Concepts
"I am a ________, and when I ________, I would be able to ________, if I knew ________."
Test
Develop and test prototypes
Solutions
Development of final solution
Deployment
Inspiration from the final solution sparks new ideas

Use case identification often looks like a classic innovation process. A wealth of ideas must be generated, sorted, prioritised and enriched until some of them can be tested, and finally a subset can be sent out into the world for feedback and inspiration.

Idea generation

The first rule of data science use case identification is: **Never do it alone**. Get out there and meet the potential users. Conduct workshops or invite people in. Start by explaining what data science and machine learning is followed by a question that can help with inspiration:

IF YOU HAD A MAGIC CRYSTAL BALL THAT COULD ACCURATELY PREDICT THE FUTURE, WHAT WOULD YOU ASK IT ABOUT YOUR WORK LIFE?

After the initial jokes about lottery numbers and the private lives of their friends, it often turns out that that question is harder to answer than anticipated. It requires that people think quite specifically about the question which they would ask the magic crystal ball. The great thing about the exercise is that it puts discussions on hold about what is possible and what is not and allows people to be creative. Often, many interesting questions and discussions come up, many of which are obviously impossible, and this is part of the point. It is much better to discard an impossible idea early on than to develop something realistic but worthless.

Creative enrichment

At this point, we would like to have 20-40 ideas written down which describe situations where knowledge from a magic crystal ball could be useful. This needs to be further clarified, and we need to be critical of our issues. We know from the previous exercise that the ideas are **interesting**, but are they also **valuable**?

The purpose of this step is to validate that the problem we are solving is actually valuable. That is why we use this phrase:

I am a ____, and when I ____, I would be able to ____if I knew ____.

The key is to have an action in a situation. See this example:

I am an operator at 112, and when I make a call, I would be able to send an ambulance faster if I knew it was a cardiac arrest.

(See (Corti.ai 2020) for more on this example)

If such a sentence cannot be constructed, there is a risk that what has been identified is a "nice to know" type of problem. A situation where someone would like to know something because it could be nice or interesting but may not actually be able to use the information for some valuable action. These sentences also ensure that there are concrete people to go to in order to get the idea verified. Do these people actually find themselves in the situation described? Is the information proposed enough to make that decision/ take action? Is the action actually available? If the answers to these questions is "yes", then we have an enriched idea.

Tip: If possible, ask the person mentioned in the sentence if they know of any data that could give a clue as to what we would like to know. People are often very creative and can have insights that we do not have so take the opportunity to talk to users!

How about complete automation?
In a complete automation situation, one in which there is no human to consider in terms of information and taking action, a solution might look different. This type of use case falls outside the scope of this section. Not because they are not interesting, but because the identification follows a somewhat different pattern and lends itself more easily to a classic cost-focused business case.

Critical questions

Ideally, we should now have a handful of good concepts to move forward with, and it is time to look at the data science feasibility behind each. I often use these three rules of thumb:

- How long would it take a human to answer the same question?
- Would ten people be more or less in agreement or would disagreement be expected?
- Does data exist which we can reasonably expect to explain the phenomenon?

How long would it take a human to answer the same question?
This question is about the expected complexity of the problem. An old rule in data science says: if it takes a human more than two seconds, the computer probably cannot solve it. Today, we have come somewhat further, and I would be willing to say "a few minutes", but the principle is still right. If people find it to be a difficult assessment, the machines are probably least as challenged. The best use cases are often easy for people. Assessing whether an image is of a dog or a cat is easy, and we do it almost without thinking about it. Conversely, assessing whether the content of an article is factually correct may require us to think carefully.

Would ten people be more or less in agreement?
The question attempts to assess the degree of subjectivity of a given problem. Note that the world is full of things that we know to be subjective, but there are even more things that we think are objective, but which in practice are not. Credit assessments, grading, various applications and requests. All of these areas are difficult for data science because they (for good and bad) contain a lot of subjectivity.

It is worth noting that subjectivity is not a dealbreaker for data science. If a task is subjective, it is cause for awareness for the data scientist and respectful humility as to what can be expected of a solution. Algorithms can also be used to identify and control bias. This will be elaborated on in Chapter 5. Ethics.

Does data exist which we can reasonably expect to explain the phenomenon?
Some phenomena do not lend themselves to prediction because they are fundamentally unpredictable. Earthquakes and volcanic eruptions have long been the goal of science in terms of trying to understand and predict them but so far without major success. Major political events such as elections and civil wars are also notoriously difficult to predict. The stock market is also a classic challenge. Common to all is that we could probably compile some data, but we would hardly be able to compile enough data to obtain a particularly strong model, either because data is particularly hard to come by, and/or the phenomenon is known to be incredibly complex. The best use cases are those where the source of knowledge is very clear. If we need to diagnose if a leg is broken, we take an X-ray. We know that the answer to our question ("Is the leg broken?") can be found in the image. We might just not know exactly how. This is where machine learning is strongest.

Test and development

Hopefully, you have at least one idea left after all of this. Congratulations, you probably have a great use case! You are reasonably sure that there is a group of users who, given your solution, could take valuable action with it, and you are reasonably sure that, realistically, it can be developed using a data science approach. It could now be extended into a formal business case with estimates of effort needed to develop the solution, expected business impact and prerequisites and requirements for development.

All of this will be further elaborated on in the coming chapters.

Summary

If you need to identify valuable data science use cases, it is helpful to first initiate a creative process by putting the limitations of the technology on hold and focusing on identifying situations where better decisions can be made based on information. It can be done, for example, with a phrase like: I am a __, and when I __, I would be able to __ if I knew __.

Typically, such phrases can be addressed by a data scientist who, based on a critical assessment, will be able to prioritise them and begin collecting data.

Chapter 2. Definition of purpose

If there is one thing I have learnt during my time as part of Implement's data science team, it is the importance of a clearly defined purpose all the way down to the model specification. Data science is relatively unique in that goals and purpose are ultimately formulated in explicit mathematical equations called loss functions. In short, it is the art of ensuring that our data science solution actually solves the valuable problem which we have identified and not a simpler (but related) problem, and that it actually happens under realistic conditions. If we are not intensely conscious of our loss function, we risk, for example, prioritising our customers incorrectly, or we risk optimising for situations that do not arise.

So far we have discussed what data science is, how we identify a valuable problem, and how we make sure that we have the data we need to solve our problem. The next step in the process is to define the goal and the boundaries of our problem.

- What is a loss function?
- Why are loss functions important to the business?
- Tip #1. Weights
- Tip #2. Not all errors are equal
- Tip #3. Don't try to do the impossible
- Baselines

What is a loss function?

When training a machine learning model, we often set up a loss function. The purpose of a loss function is to describe mathematically to the model what we are trying to optimise for. Typically, we deal with three types of problems in machine learning:

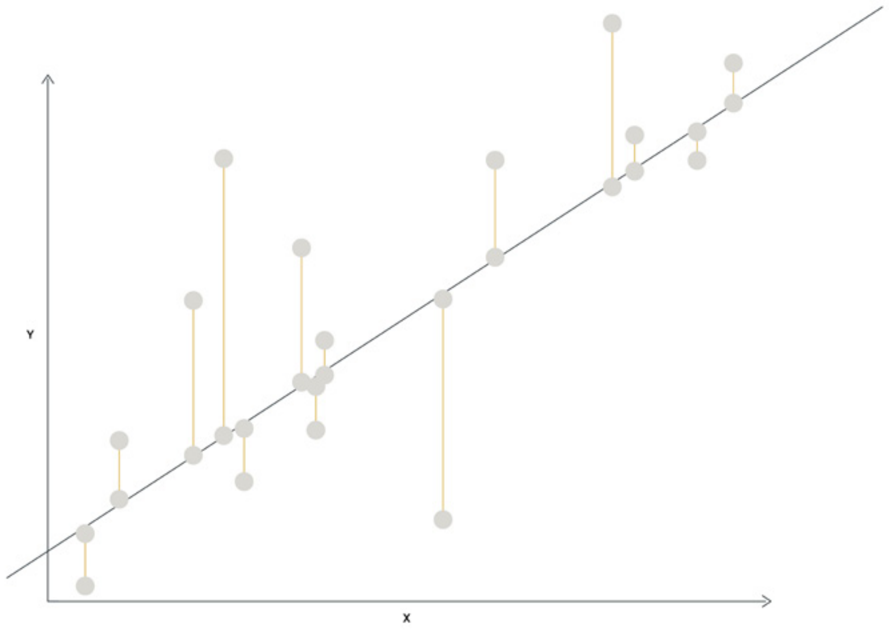
1. **Regression:** Guess a number, for example the price of a stock.

2. **Classification:** Guess a type out of multiple possible types, for example the diagnosis of a patient.

3. **Binary classification:** Yes/no questions. For example: "Is this spam?"

For each of these types of problems are some canonical formulations of loss functions. For example, to the right is an example of regression. The diagonal line is our model, and the dots are our observations (i.e. the data). We want a line so that the distance between the line and the points is as small as possible for all the dots.

For classification, there are several variants, but a natural solution is simply to look at the proportion of correct answers and try to estimate an algorithm that minimises the number of incorrect answers.



Why loss functions are important to the business

It is often surprising to people closest to the business that when we make regression models (i.e. guess a number like for example the number of goods sold tomorrow), the standard practice among data scientists today is to minimise the average error (as mentioned above) but to weigh more errors exponentially heavier the larger they are. This formulation has some appealing mathematical properties, but it is a very important business decision that often just slips through. In practice, the business often has strong opinions. Maybe errors below a certain size are truly insignificant, or maybe errors above a certain size are all equally “bad”. If this is done wrong, a data scientist may mistakenly develop an algorithm that is rejected by the business, because the algorithm optimises too much for some extreme periods (or worse, non-existent scenarios!) at the expense of everyday operations. On average, the model becomes too imprecise, and the project is abandoned. That, although a proper wording of the loss function would have solved it.

Tip #1. Weights

Ask the business early on: “Are there certain things more important to identify than others?” The answer is almost always yes. Maybe finding the cancer cases is more important than avoiding taking in too many patients. Exactly how much more important? This is an important business question that needs to be answered. For example, a project in Denmark found that it was 16 times more costly to overlook a “yes” that should actually be a “yes” than to say “yes” to something that should have actually been a “no”. When asked the question, it actually appeared that there was a clear business calculus that could provide the exact answer. Obviously, it dramatically changed the algorithm compared to if “yes” and “no” had been equally weighted. Most loss functions allow weights to be applied to individual cases during development to allow the model to focus on the most important problems.

Tip #2. Not all errors are equal

Commonly, the size of the errors is also not of equal importance.

Errors so small that they don’t matter: Depending on the solution’s use, a +/- 1% error may be negligible. Then it might be helpful to just ignore them in the loss function. Techniques such as support vector machines make this very explicit.

Errors so big that they don’t matter: An even more important category is the big mistakes. Maybe obviously, large errors are easily caught by humans or are possible to check using rules. It might even be preferable for a solution to provide on an obviously false answer than deliver something close.

Between these two filters, the model gets a chance to focus on the cases where it can make a valuable difference. The cases can be sorted from data (here you have to think carefully!) or implemented as part of the loss function.

Tip #3. Don’t try to do the impossible

If a problem proves particularly difficult, the introduction of a “don’t know” or “grey” category can make business sense. In a classification problem (“Is it ...” question), this is often done by taking a large number of rare types and adding them to one large “grey” class, which is then handled manually or according to some known rules. In regression (“what is the number” question), it can be done, for example, by first estimating a model that tries to estimate the exact number, after which you inspect the cases in which the model makes big mistakes. Next, an individual model is trained to assess whether a case is likely to have a large or small error. Finally, the original model is only retrained on the proportion of cases that had a small error (and the errors are then re-examined, and we would expect to generally see better performance). When the model is to be used, the "big or small error" model is used first. Cases with expected large errors are handled manually, while only those with expected small errors are allowed to pass.

For a more comprehensive overview of different models, I recommend scikit-learn (scikit-learn 2020).

Baselines

A baseline can be whatever the current solution is (or a simple way to develop a rule). Maybe it is just a rule (we just always say yes) or purely human judgement. Perhaps this is due to some simple heuristics ("We know that our employees generally say the same thing as yesterday +/- slightly depending on these known conditions").

Baselines are hugely important. The business is rarely interested in knowing that our model is right 84% of the time in isolation. They want to know how much better than the current practice we are – the so-called **"model lift"**. (Or how close can we get to the human baseline with an algorithm that can work for free 24/7?)

Model lift

A concept of how much better a model is compared to a baseline. It may be tempting to focus solely on the precision of a model, but in practice it is often the improvement over the starting point that is the deciding factor.

Summary

Before we start selecting and designing models, we should make it clear what the purpose is right down to the mathematical level. Are all types of errors equally important? How good is the baseline we are up against? What is the cost of making a mistake versus sending a case for manual processing?

Those questions are at the heart of designing the loss function and purpose of a data science solution.

Chapter 3. Data exploration

Data exploration is probably one of the most underappreciated data science disciplines. You should not underestimate the time you can save by getting a strong feel for your data before you start modelling.

You can save a lot of time by knowing your data well and designing more clever models and data transformations earlier while avoiding serious mistakes or bias later in the work. Since data exploration is not suitable to put into a well-defined standard set of procedures, I have instead compiled the five most important tips that I try to follow as well as some of the most important tools.

Understand the source of the data

All data come from somewhere. Whether it is manual entry, a sensor in a machine or a digital system monitoring some kind of behaviour.

It is crucial to understand the quality and context of the source. If possible, try to meet the people who either enter the data or work with the systems that create the data. What is their impression of the work associated with the data they create? Do they have confidence in the data? Are there exceptions and dark numbers (i.e. non-recorded cases) in their work?

If possible, follow the data from the source to the point where you expect to access it. Are there business decisions along the way that you need to be aware of?

For example, you can ask the source or source owner the following questions:

- Would you trust the data you provide yourself?
- Are there things that are not captured and expressed in data?
- Is there any data that will not be delivered?

Ask local data experts

Now you understand the data sources, but it can still be important to ask experts in the respective data areas.

If you use data from multiple sources or make transformations of your data yourself, make sure that the transformations you make are known and approved by people in the field. Do you retrieve labels from a table and link it to some data in another system? Does that link make sense?

You should also take the opportunity to enquire about data availability. If the problem you are trying to solve is time-sensitive, is the data updated frequently enough to make sense for your solution?

For example, you can ask the following questions to the data expert:

- Can I link these sources? Is there anything I need to be aware of?
- How long is the history of the different sources? Is it consistent and reliable over time?
- How often is data from the sources updated?

Keep track of your data structures

If you are to take one thing away from this, it would be the following: If possible, set up an Analytical Base Table (ABT).

An ABT is a very safe and easy way to handle your data in a data science project, and it is hugely suitable for data exploration. An ABT is basically a large table that has all your observations as rows, and the columns are the properties that we know about our observations.

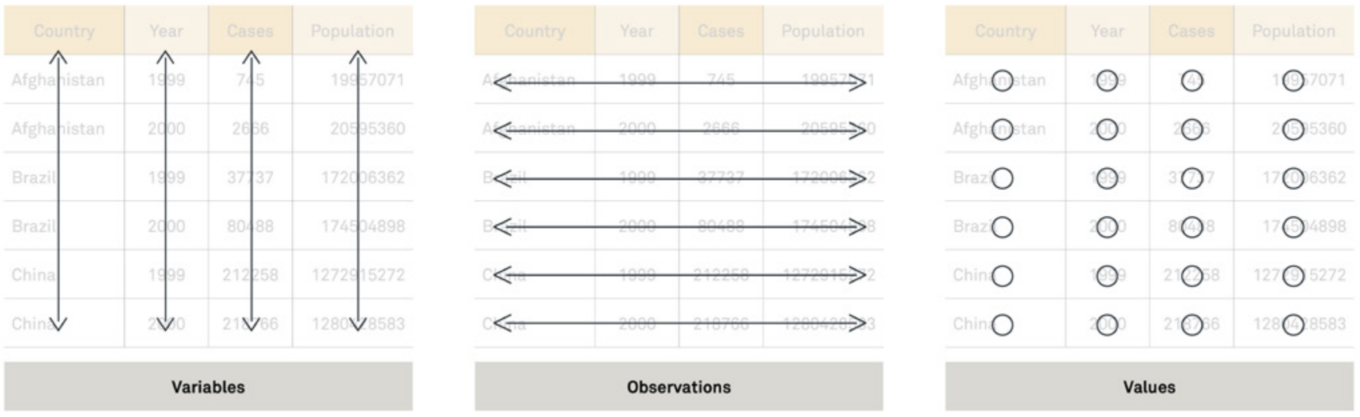
Analytical Base Table (ABT)

An ABT is a table that strictly adheres to the principles of tidy data. Thus, an ABT has one row per observation and one variable per column. Among the columns are one or more "targets" that are what we want to predict or analyse, and one or more "features" that are what we know about a given observation that we should use as a source for our predictions or analyses. It is a simple and very effective data presentation that makes the use of a wide range of data science solutions very intuitive.

You build an ABT in the following way:

1. **Establish a base population.** What are you analysing? Are these support cases? Process steps? Customers? Or customers per day? This question is harder to answer than you might think, so think carefully. When you can answer exactly how many observations your base population has (hint: it must be a counting number), you have your answer.
2. **Left join features onto your base population.** A left join ensures that observations from your base population (your left table) are never removed when you add new properties from a source (the right table). Instead, missing data is inserted if there is a discrepancy between your base population and your source. It is an important safety feature, so that you do not suddenly lose data. At the same time, it is easy to check if duplicates are triggered. You can always count the rows in your base population, and it must not be larger than when you established it.

If you have done it right, you have so-called tidy data (Grolemund og Wickham 2020). Each observation is in a row, each property is in a column, and each cell is a property for a given observation.



Country	Year	Cases	Population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Tidy data

Tidy data is a data structure that we always strive to adhere to in all data science tables. Tidy data has one row per observation, one variable per column, and one value per cell. Thus, it is a flat two-dimensional table. It is far from always that we can keep our data tidy when analyses and needs get complicated, but we should always strive for that. As long as most of our tables are tidy, we are in a much better place.

Visual inspection and tools

If you have established an ABT (or something similar), you can do a visual inspection of your data. It is important to look at data. Visualisations are much richer than lists of, for example, averages, standard deviations etc. If you want to see a terrifying example of how descriptive statistics can lie, take a look at Anscombe's Quartet (Anscombe, 1973) or, even worse, Autodesk Research – Datasaurus (Matejka og Fitzmaurice 2020).

Below are three excellent free tools for data exploration:

- **Google Facets:** A very simple and fast data profiling tool. You upload the ABT (or download the tool and run it locally) and get quick descriptive statistics and simple distributions across all your key features. However, the real value comes when you cross variables. In Google Facets, you have all the data in a simple interface, making facets the easiest visual inspection tool (Google 2020).

- **Tensorflow Projector:** A substantially more specialised tool designed for complex data with a lot of inter-dependent features. Tensorflow Projector can "distil" complex data-sets into 2D and 3D visualisations, grouping similar data points together (Tensorflow 2020). The technique is called T-SNE (Wattenberg, Viegas and Ian 2016) and is a powerful visualisation algorithm. I personally use Projector in two ways:

  » To ensure that there is a good and varied structure of data. If Projector makes an artificial or unnatural looking form, it is often because the structure of data is not as rich as we might otherwise like it to be.

  » Projector is uniquely suited to handle extremely complex data such as text, audio or images. Once it has been processed and prepared for analysis, it can give a human-friendly overview of the dataset.

- **Microsoft Power BI:** In all fairness, Power BI is a complete business intelligence tool, but the fast calculator and powerful universe of visualisations make Power BI an important data exploration tool. The tool is especially suitable because you can make regular reports in Power BI, so without additional effort you can have dashboards to monitor data quality as you develop. In addition, Power BI is particularly suitable for handling geographical data (Microsoft, Power BI, 2020).

Automation

I highly recommend setting up automation around your data exploration, and especially data profiling, so that it happens often and in a reproducible fashion. Packages such as, for example, Pandas Profiling (Profiling, 2020) for Python make it easy. The whole point is to make it easy to do data exploration so that this discipline is kept up and you get it done.

The data profile can be very basic but make sure that you always keep track of how many observations, how many – and which – properties, what is the minimum and maximum of each attribute and how many missing data points each attribute has.

Furthermore, automation has the advantage that it can be done through an analysis flow. It is a great help to have consistent data reports for all sources for the overall ABT and for all significant transformation steps towards the final model and final output.

These reports are also an essential aid in handover and maintenance, helping others to understand the data that you have been working on.

Chapter 4. Machine learning

Machine learning is the process of having an algorithm that learns patterns in data by itself.

For example, a basic example could be an algorithm that takes a series of numbers (x) and multiplies them individually by one number (a) and adds one number to (b) trying to get the series to come as close to another series of numbers (y) as possible. So a formula of the following type:

$$a * x + b = y$$

This is also known as linear regression. We know from high school that we can work out a solution to this, but there is another solution too. How about taking a fast computer and trying 1,000 different numbers in place of (a) and 1,000 different numbers in place of (b)? Then simply calculate which combination of (a) and (b), on average, made (x) closest to (y). On a modern laptop, it takes well under a second and gives a result that is very close to the exact mathematical solution. We can improve it even further by using an algorithm that looks at how different combinations of (a) and (b) give a more or less correct answer and use that information to make either (a) or (b) bigger or smaller. With that, we can get a satisfying answer in as few tries as possible.

For simple linear regression it might be silly because we have the good, exact mathematical solution, but for many statistical models (several are mentioned here), there is no exact mathematical solution. Instead, we have to rely on approximation. Thus, solutions like the one above where we optimise our guesses can actually be our only option. It is just our luck then that it turns out that such optimised guesswork performs so incredibly well.

So now your data is ready, and the problem definition is in place. It is time to estimate the right model:

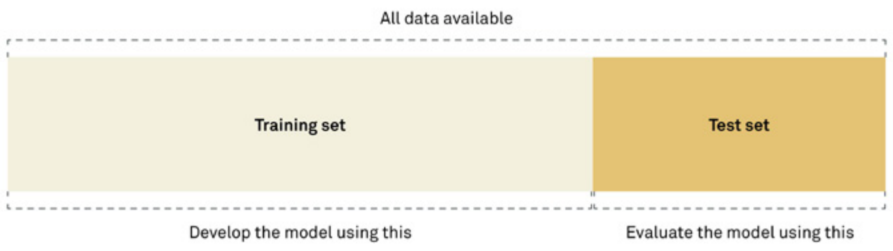
- How do we identify the right model?
- Overfitting and underfitting
- What to do about fit?
- Classical models
- A warning
- Bonus: Cross-validation

How do we identify the right model?

The right model is the model that does what we would like it to do in the future, i.e. on new data. For example, it could be 100,000 e-mails, of which 10% are spam, to which we would create a model that could act as a filter on all future e-mails against our e-mail server. This kind of formulation might be obvious, but it has important consequences.

The classic way to do this is with a test-train split. The principle is quite simple. We take our data and split it into two chunks. One for "training" and one for "testing".

We then estimate our model on training data alone and try to use this model to guess our test data. The charm is that we know the truth about our test data, so we can measure very accurately on how accurate the model is.



Take the example of the 100,000 e-mails. We would take 20,000 e-mails randomly (spam is still around 10%) and save it for testing. We would then "train" a model of the 80,000 remaining e-mails by showing it the contents of the e-mail – spam or not. When we are happy with the model's ability to predict whether an e-mail is spam or not in the training set (maybe it is right 95% of the time), then it is time for the exam! We then take the model and test it against the 20,000 e-mails we saved. On the test set, it may hit right in 91% of cases. Then we can assume that we have a model that is in fact correct about 91% of the time.

Overfitting and underfitting

The reason we have to do this test-train split is because modern machine learning is really good. So good in fact that with a sufficiently advanced model (more on that in a little while), I can promise that any data scientist can guarantee 100% accuracy on training data. The most advanced models can completely "memorise" large datasets, but they do not learn anything useful in general about the data. The phenomenon is called overfitting and is absolutely crucial in the quality of a machine learning model. This can be illustrated as below.

Overfitting and underfitting

Overfitting and underfitting are the two most important phenomena to balance when building models. Overfitting means that our model is too complex and learns nuances in data that have nothing to do with the real world but are about specific patterns in the examples we have. Intuitively, one can say that the answers are learnt by heart without learning the contexts that give the answers. Underfitting is the opposite where the model does not have the sufficient complexity to capture the patterns that are in the data. Intuitively, it can be said that it is not smart enough to learn the information available in data.

The below diagram shows the essence of the challenge. The greater the model complexity, the better the model will "fit" on training data and initially test data (because there is complexity to capture). But after a certain point, learning starts to go awry. The model gets better and better on training data but begins to "over-interpret" data. When it later encounters new data (like the test set), it suddenly creates a lot more errors.

The phenomenon is illustrated in the three figures above. For example, the complexity of the model can be seen as how "complicated" a line we allow the model to draw through our data points. In the first figure ("Underfit"), we only allow the model to draw a straight line. Here, the model is just bad. It does not fit very well with data, and both testing and training have many large errors. In the next figure, we allow the model more freedom. It must not make sharp turns, but it is allowed to bend the line. Now, we can see that it follows data really well, and we can assume that if only future data follows roughly the same structure as our training data, the model will probably do an excellent job on it as well.

In the third and last figure, we see the overfitting scenario. Here, the model has been given too much freedom, and it has started to simply connect the dots. It is probably 100% accurate (it hits all the dots), but if we put in a new dot, the chance that it would be particularly good on the line is very small.

What to do?

There are a few classic tricks in these scenarios:

1. **Test often!** Vary the parameters in your model and see how the difference between testing and training changes. Keep increasing the complexity of your model until you see no improvements or the improvements are only on training data.

2. **Support the model with additional information!** Also known as "feature engineering". This is a manual process in which we try to help the model by removing distracting information and highlighting important elements. This obviously assumes that we know what data are merely distractions and which are important. It requires strong domain knowledge. In our spam e-mail example, a classic trick is to throw away words like "that", "and", "it", "there" and "are" because they are not important to the content. This means that there are fewer words left to distract the model.

3. **Get more data.** Overfitting is always relative to a given amount of data. Keep in mind that the challenge is that the model "cheats" by learning data by heart instead of the general patterns and intuitions in the data. The best solution is actually to just give the model more data. With enough data, even a highly complex model can no longer "remember" everything and is instead forced to learn more general and useful properties instead.

Feature engineering

Feature engineering is when a data scientist manually performs mathematical operations to help a model see the relevant patterns in data. For example, in a situation where we want to assess whether a credit card transaction is fraudulent, relevant features to engineer could be average transaction size for the cardholder, average transaction size for the seller, the number of times the cardholder has purchased from the seller in the past and how long since the last purchase. Information like this would probably be very helpful to our model, while the name of the seller would probably just be a distraction.

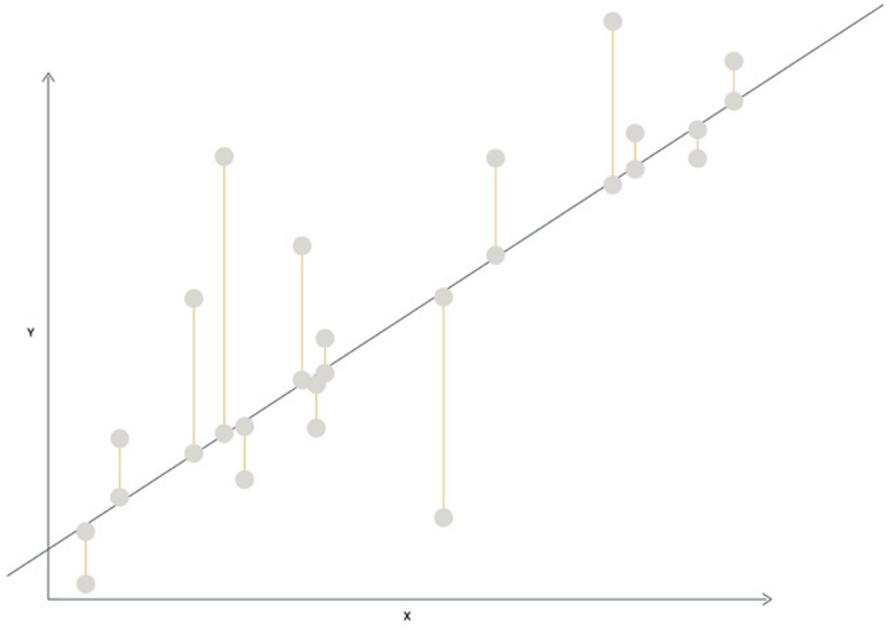
Classical models

There seems to be an inexhaustible range of data science models, all of which are ideal within their own little niche. That being said, there are a handful of classics that everyone should know, and that can often solve 99% of problems adequately.

Logistic and linear regression

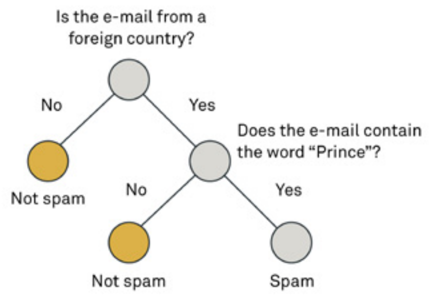
Classic linear and logistic regression are still some of the most widespread and helpful models today. Both models work by trying to estimate a given target (e.g. the price of a stock tomorrow) based on an input (e.g. prices over the past seven days) and then fitting a line.

The fit is obtained by estimating the line which most closely fits the inputs to the outputs, like in the example in the beginning of the article. The figure below illustrates the model.



Logistic regression is a bit more complicated, although the principle is the same, but here the line can take other forms than a straight line. For example, it may take the form of an S-curve.

Decision trees and random forests
A less famous alternative to linear regression is tree-based models. They are called tree-based because they are all based on a so-called decision tree. For example, a (very simple) decision tree to determine if an e-mail is spam might look like this:


Is the e-mail from a foreign country?
No
Yes
Not spam
Does the e-mail contain the word "Prince"?
No
Yes
Not spam
Spam

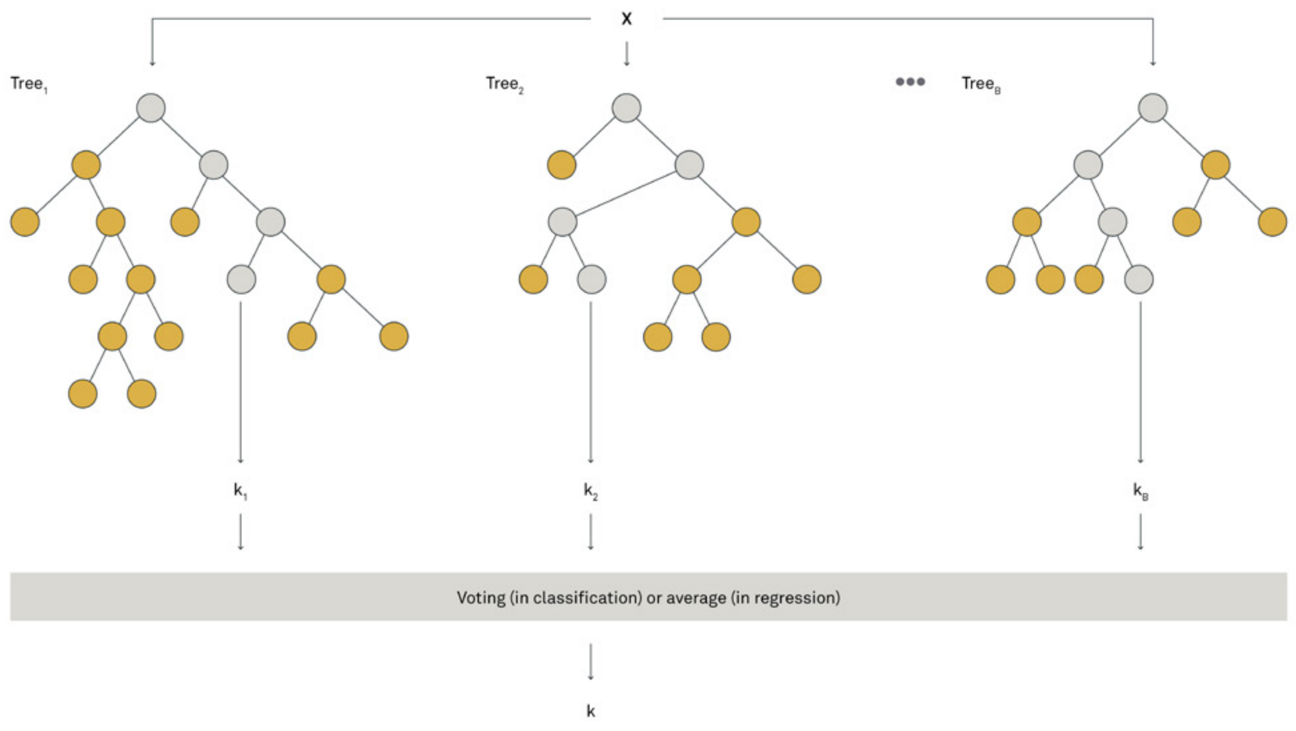
There are techniques for learning that kind of decision tree from data by constantly asking: *"What do I need to split on now, so that the groups I split into are most pure in terms of what I'm interested in (e.g. spam/non-spam)."*

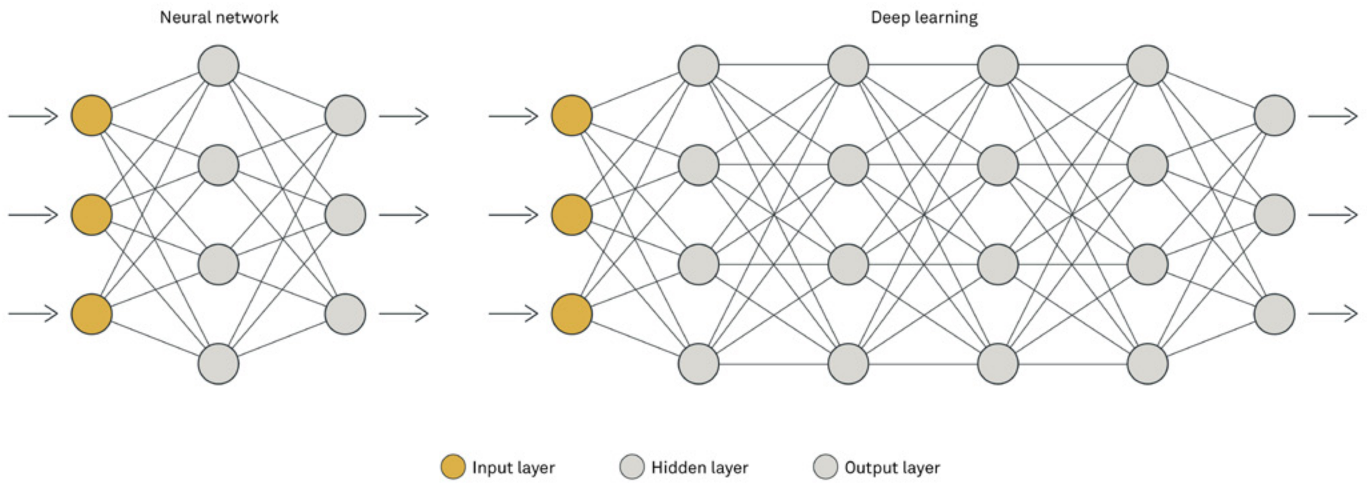
A development on the decision tree is the random forest. Here, we take a sample of our data (e.g. 20% of rows and 50% of columns) and estimate a small decision tree based on this. Then we take a new sample of data and make a new decision tree. We continue this process until we have, for example, 50 small trees.

When we want predictions, we ask each tree and let them vote (or take the average of their values).

Exactly why this is such a great model is a mathematically complicated story, but random forest handles the aforementioned overfitting problem much better than a single decision tree. Intuitively, the key is that no single tree is able to know everything, making it much harder for the model to merely memorise everything. Instead, each tree "does its best", which is alright in itself, but with a lot of them, rather accurate estimates can be obtained. Additionally, random forest has a lot more settings to tweak like: How much data does each tree have? How many trees in the forest? How complex is each tree allowed to grow? We have a very natural way to increase complexity (like adding trees to more data to each tree).

Gradient boosting is a further development from the random forest which is worth mentioning. It is a rather complicated model in which each tree knows the errors of the previous trees, meaning that the trees are allowed to support each other and specialise in certain nuances of the data. It is an extremely powerful method known for winning a lot of smaller data science competitions on the web.



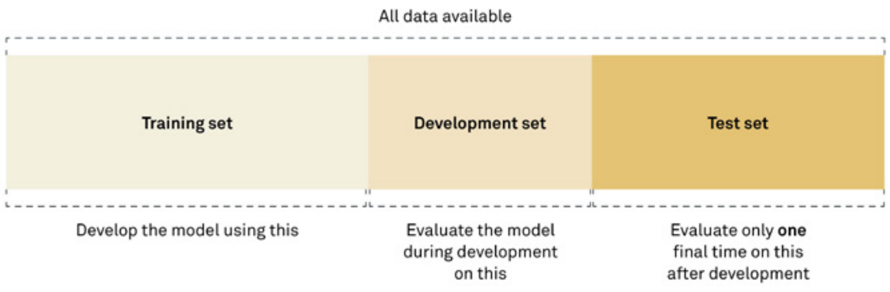


Deep learning
Deep learning is probably one of the hottest techniques right now and is the driving force behind almost all major artificial intelligence breakthroughs in recent years. The technique has become very advanced, and there is a whole field for them which is why it is almost unfair to describe just one technique.

In short, deep learning works by "stacking" mathematical functions on top of one another with non-linear transformations in-between. If that did not make sense, think about it like this. Maybe you have three "inputs" for your model (height, age and income), and you want to guess what gender people have (male/female). In a deep learning model like the one below, you take the three parameters, cross them in a number of ways (four ways in the illustration) and then transform them through one of the different "neurons" (the first layer after input). Each neuron has a non-linear transformation, so the output is between 0 and 1, and high and low numbers come closer together, while numbers around the middle spread out between 0.2 and 0.8 (this is also called a "squashing function", as it squashes the numbers into a given range). The four new outputs are then combined with each other in the second layer of the model, whereupon they are transformed through new non-linear functions, then onto the third layer etc. Finally, they reach the last output neuron, where a number close to 1 means female, while a number close to 0 means male (and when we use it later, we would set a cut-off point at 0.5). When training the model, weights are identified for all inputs between all layers (so that inputs from the previous layer are not just naively put together). These weights might increase or decrease the signal from the previous input. The specific properties of the non-linear functions are also estimated (when should they begin to "squash" – at 0.2 or 0.3?). It quickly becomes very complicated, and therefore deep learning is an extremely complex method. The advantage is that it can capture even the most incredible nuances in data. This is why it is often used in, for example, languages and images that contain a lot of information and complexity. The disadvantage is that they are very prone to overfitting and are therefore extremely data-intensive.

A warning

Above, I have described a process where a single chunk of data is taken aside for testing while training on the rest. If this goes on for a very long time,


All data available
Training set
Development set
Test set
Develop the model using this
Evaluate the model during development on this
Evaluate only one final time on this after development

and a large number of model types are tested, the process of selecting a model can be driven by one's test set rather than something real in the data. Many people therefore work with an extra split, a so-called development set.

Basically, data is divided into three sets: 60% training, 20% development and 20% testing.

Training is used as before to estimate the model. Development is used to "test" on during development of the model (it is used multiple times). The test set is now used only once – as a final exam.

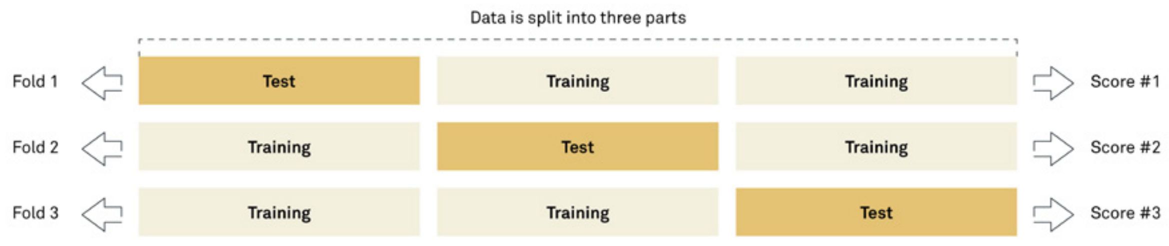
This ensures that the test set really is new data, and that the precision we report for our model will be on the final set.

Bonus: Cross-validation

An alternative to the test-train split is cross-validation. Data is split into a number of parts (for example three parts with 33% data in each). The model is then estimated and tested once for each chunk of data – each time with a new chunk for testing and all others inside as training kits. So for each "fold", a test score is obtained, and we then calculate an average across all folds as our final score.

Cross-validation is generally considered both more accurate and safer than a simple test-train split, but it can also be time-consuming and complicated to implement because the model has to be estimated many times.

Note that it is often recommended to split data into 5-10 pieces when doing cross- validation. This ensures that the test set has a reasonable 10-20% size each time, but once again increases computation time by an equal factor.



Chapter 5. Ethics

In the field of data science, ethics is very important at the moment and for good reason. When we work with data science, we often get very close to people, and we use tools that we do not always fully understand. The field of data ethics and data regulation is experiencing about as rapid a development as the field of data science, and we see legislation from both the EU and other actors trying to set some frameworks specifically around the use of artificial intelligence and data collection.

Within moral philosophy, ethics is the discipline that deals with the question of what is good and evil or right and wrong. Here, however, I will specifically deal with applied ethics, which are questions about what a person is required to do (or what is allowed) in specific situations.

I should note that what I write here is very much my own take on ethical data science. I have studied and worked actively with ethics both in general and in data science for several years. In addition, I lean on People and AI Research (the PAIR group) and their work with the People + AI Guidebook. However, I have not contributed philosophical breakthroughs myself, and I cannot claim to systematically rest on an established moral-philosophical system.

Privacy

I want to go over privacy issues fairly easily here. It is not because it is not important (it actually is really important) but for the sake of space and focus in the guide. For a comprehensive insight into the privacy discussion, the UK Information Commissioner's Office (ICO 2020) is a great place to start. Privacy takes other forms beyond the purely legal aspect like GDPR. Fundamentally, as with a lot of applied ethics, it is about finding the balance between the good we are able to do, and the risks and actions we have to take to do it. The discussion is most fruitful, when it centres around the things we can do best to allow people control and agency in out solutions, like allowing people to see their data and obtain some kind of explanation behind an algorithm decision. It is also fruitful to discuss technologies which might aid in the safety of solutions from basic things like encryption to advanced solutions like differential privacy, where data is added a little bit of wrong information (so-called noise). That is, if data were to fall into the wrong hands, it is not known which is real and which is false. Adding the false information correctly will not affect the data science value of data.

A very basic way to view things is with a simple dos and don'ts of ethical data science:

Dos:

- Create value
- Prolong life
- Make people's lives easier
- Make expertise available for everyone
- Automate tasks to free up time for people to do more valuable things

Don'ts:

- Bias (here meant to have a systematic inaccuracy in the model)
- Discriminate (here meant as the specific kind of bias related to protected groups like gender, race and religion)
- Obscure or manipulate
- Lie

Make sure what you do is valuable

When we consider the justification for our data science solutions, it is important to assess both the good and the bad. Data science has tremendous potential, and we have examples of great good being done like AI saving people's lives (Corti.ai 2020), making people's lives easier (Muhajlovic 2020), making expert intuition available for more people (Seeing Potential 2020) or automating tasks to free up time for people to do more valuable things (Enversion, KAUNT 2020).

AI and data science create – and must create – a lot of value. If a solution is not valuable, it can hardly be justified under any circumstances (DeepNude, (Vincent 2019)). There is a stark difference between mining patient journals to advertise phony nutritional supplements to them and trying to prevent strokes (Enversion, Sundhed 2020)?

Almost all data science projects try to be valuable, and almost all of them have a firm grasp on their impact beyond the immediate profit motives. That is why it is so much more important to keep it this way.

Avoid bias, overconfidence and unreliability

The first class of ethical problems to deal with is quite statistical by nature. Bias is when a model exhibits systematic errors, where it might over-classify cats or spam or consistently predict the rise of a stock too low. Unreliability occurs when our model makes more errors than we anticipated, but there is no systematic structure to these errors.

Both challenges are a business challenge but by extension also often an ethical challenge because it means that we are potentially releasing a model that is less valuable than we expect, making any judgements about its merits questionable.

Both unreliability and bias usually occur in one of two ways: (1) We have been careless and made a mistake in our data processing, or (2) our data were poor and did not fit with the reality in which the model was to be used.

We can remedy point 1 by using a thorough method, meaning thoroughly testing the models (Chapter 4. Machine learning) and making sure that the models are actually performing as expected (Chapter 3. Purpose definition). Another important technique is to focus on outliers and edge cases and ensure elegant handling of these as well. Either by making sure that the model in fact handles this well, or that alternatives are in place (see the People + AI Guidebook, the chapter Graceful Failure for great inspiration).

Finally, finished models can be subject to simulator studies where counterfactual cases are processed through the model, and we are able to closely investigate the provided results.

Point 2 is best remedied by knowing our data intimately. Good data exploration is key (see chapter 2). Both the actual statistical structure of the incoming data as well as its origin.

I consider it to be a data scientist's ethical responsibility to ensure that his or her models are robust, and that the data they are built upon are clearly understood.

Avoid discrimination

In this context, I use discrimination to mean when a model has learnt an undesirable behaviour to the detriment of protected social groups. Classic examples of this are racism and sexism. Discrimination can be because of bias (either in data or in model development), but I will extend it even further. Even a perfectly designed model on a well understood data foundation can exhibit discriminative properties if it systematically disadvantages protected social groups.

This means that we have two types of discrimination:

1. **Discrimination in model development.** When the data is good, the question is well formulated, but some error introduced by the data scientist introduces discrimination. This is a relatively easy case, as once it is identified (through testing and data exploration with special attention and care to protected social groups), we can simply fix whatever error introduces the discriminative property.
2. **Discrimination in data.** Oftentimes, our history is undesirable, and we might have had discriminative policing or crime patterns. Cathy O'Neil thoroughly demonstrates in her book, Weapons of Math Destruction (O'Neil 2016), how cleaning data can be difficult, impossible or make things worse. In fact, she ends up concluding that it is best to include the discrimination and, instead, pay attention to it and deal with it around the model. This obviously requires a thorough study of the model's discrimination (for example with simulations of counterfactual cases).

On the other hand, when there is discrimination in data, it is often because there is an existing discriminatory practice, and as such, data science actually has an opportunity (some might even argue an obligation) to cast light on this and try to improve on it.

In the words of John Shawe-Taylor, UNESCO's (UN) Chairman of Artificial Intelligence (Synced 2019)

"HUMANS DON'T REALISE HOW BIASED THEY ARE UNTIL AI REPRODUCES THE SAME BIAS."

So before we attack a discriminatory model, it is often worth asking why data is so unjustifiably discriminatory in the first place. Is it possible to make a less discriminating algorithm than the human alternative? The ethical question is often about which of the following three solutions is least problematic:

1. The improvement is worth making despite the concern about automating discrimination.
2. The discriminatory process can be completely abolished or radically rescheduled.
3. Stick to the discriminatory practice that it is.

All three options should be considered and can be valid. In the following, I assume that option 1 was chosen, and that we have decided to develop a model and want to pursue an as ethically defensible solution as possible.

Avoid lies, manipulation and obfuscation

Even if our model is great, it is very important how we talk about it.

We can imagine a data science solution that deals with people's claims for compensation from an insurance company. We assume that the solution is relatively well-functioning and does two things: (1) Assess whether people have filled out their application with sufficient information and (2) give them an immediate assessment of their claims.

Here are three examples of good and bad ways to talk about data science.

Avoid turning data science into black magic

X *Thank you for your application. We will now process it with our Auto-Request-Analysis_AI™ and give you an answer faster than ever before!*

This response risks leaving people confused and unclear as to what exactly happened. What is this system? Why is it happening? What exactly does it do? What if I don't like it? It is unhelpful marketing hype.

Avoid hiding the model

X *Thank you for your application. We will process it and get back to you as soon as possible.*

If a user then a few seconds later receives a message about their application being faulty or an initial assessment looking poor, it will cause confusion. Was it really processed? Or is this an error? Why did it happen so quickly? If you are automating tasks, do not hide it as it will only cause concern and confusion.

Avoid explaining the technology

X *Thank you for your application. We have recently implemented a non-negative matrix factorisation algorithm that helps us sort incoming applications and identify missing features. We will with 96% accuracy be able to respond to you shortly.*

This is closely related to the previous, as it leaves most users questioning what actually happened. While having a highly accurate technical explanation available for expert users is a good practice, frontloading it for everyone is a recipe for confusion, frustration and misunderstandings.

Explain what is happening, why it is happening and what people can do

✓ *Thank you for your application. We are now processing it automatically for any missing information, and – if possible – we will provide you with an initial assessment of your claim. If you have any questions about this process, simply answer this e-mail. One of our representatives will contact you soon.*

Be honest about using an automated system. It is rarely important to people whether it is machine learning, a set of simple rules or a robot. Make sure that it is clear what the analysis is about (in this case: missing information and initial assessment) and equally important: What does the analysis depend on (in this case: the application)? It is also great practice to give the user an opportunity to communicate on the occasion of automation. Most data science solutions sometimes make mistakes and by giving the user room to help deal with these mistakes as well as a chance to respond to the results in general can both help users gain necessary agency as well as provide important feedback.

Final notes on model communication

I believe that a lot of the ethical challenges in data science can be solved through good user experience design (UX). If you make sure that your users/citizens/customers understand what is happening, give them the ability to intervene and provide feedback and ensure graceful handling of failures, you have a very good opportunity to inspire the kind of trust in the model that is the foundation for ethical development. For additional inspiration, see the chapters Feedback + Control, Errors + Graceful Failure and Explainability + Trust in People + AI Guidebook.

Chapter 6. Deployment and governance

Before a data science solution can create value, it must be used. In most cases, this means that the solution has to be put into production so that it can be used continuously when needed or via daily runs on batches of data. This transition from proof of concept or prototype into production and value realisation is often called deployment.

What is deployment and governance

Deployment is the act of putting something (like a data science solution) into production. That is to make it available to either the internal or external users who need it. Ideally, the action itself is often as easy as pushing a button, but getting to a deployment and governance setup where deployment is easy is a great challenge. And because deployment marks the shift from development into production (and thus value creation), it deserves every data scientist's utmost attention.

Governance is all the processes and actors up to and around deployment, maintenance and operation. Who is allowed to submit a new solution to production? What are the requirements they have to live up to? Governance is also about the processes that ensure that errors can be traced back to their sources.

Why data science governance?

Governance has three purposes: future development, maintenance and security.

Future development: Most organisations have a complex IT landscape meaning that any and all deployments affect or depend on at least a handful of other systems (and believe me, a deployment rarely comes alone. You are going to want to update your model!). Among other things, governance ensures that the existing systems are known and that any changes to them can be made as smoothly as possible by knowing who owns and controls them and who relies on them further downstream.

Maintenance: Errors will happen. The question is how serious, how often and how difficult they will be to correct. A good governance model ensures high quality through testing, and when errors occur, they can be traced back to their sources.

Security: Governance is also the structure that ensures that there is a uniform and well-thought-out protection of data and models.

An organisation with good governance will find that they have a smooth development and deployment process, that their solutions are easy to operate and update, and that they have a strong security profile.

Bad governance, good governance

Yet, for most, governance sounds bureaucratic and heavy. This is probably because most people have been on the receiving end of bad governance.

Good governance is about having an effective, fair and strategic system.

An **efficient system** meets its goals (future development, maintenance and security) with the least possible cost in terms of time and resources but also complexity and bureaucracy. For example, there must be the fewest possible actors within each decision (which still ensures high-quality decisions and correctly placed responsibility).

A **fair system** is proportional, meaning that rules and control follow the criticality of a system. Furthermore, it is consistent, so that similar actions and problems are treated equally. Finally, the system must be **strategic** and provide support for the overall objectives of the data and/or business strategy.

And finally: Absence of governance is also governance.

A framework for data science governance

Data science governance is about creating a disciplined and systematic structure to engaging with the three main areas of a data science solution: **data, model** and **code.**

One could also add infrastructure (servers, cloud providers or other hardware) or organisational governance (do users need training?). For the sake of simplicity, I will focus on the three core disciplines in this chapter.

The exact systems that require focus can vary from organisation to organisation, but a general place to start could be:

- **Ownership and responsibility:** Who decides (and therefore has the responsibility) over which parts of a solution?
- **Processes:** How do we treat this area? What are our procedures? What is best practice? Do we have an operating model?
- **Quality:** What are the quality criteria before we deploy? A given precision? Tolerance of false positives? A degree of thorough testing?
- **Security:** Who has access to what? Are we monitoring for suspicious use of our solution? Are there any potential security holes in our code?

In short, the goal is a matrix structure like this:

Data science governance in practice

When we design our governance structure, the key is to develop the key questions and then agree on their answers. There is no exact formula to this. The below example is a great starting point and source of inspiration, but always consider your own organisational situation.

Data science governance: The data perspective
Data governance (Knight 2017) is a big field in itself. Ownership and responsibility are extremely important, and a formal role is often used to hold the responsibility for a data source, a so-called "data owner", and these are often assisted by "data stewards".

The benefit of formal roles is to decouple the function from the person, which both enables one person to have multiple responsibilities (if such is desirable) or to share functions between different individuals as needed. I can only support these kinds of concepts. Data lineage is also known from ordinary data governance, but becomes especially important in data science because we often move data around a lot. Ideally, we take note of all the transformations in terms of when they happened, what the transformations were called, how much data came in and how much data came out. This enables error tracking from a faulty system action potentially all the way back to the data source (and by extension, the data owner).

Automatic checks on data consistency should be used to sound alarms if data varies in unforeseen ways, or if the pattern of prediction changes.

	OWNERSHIP	PROCESSES	QUALITY	SECURITY
Data				
Model				
Code				

Finally, there is data security, which is handled with, for example, correct access and attention of the respective data stewards.

Key questions:

- Who has the responsibility and ownership of this data?
- What sources do the data consist of?
- Which transformation has this data gone through?
- Has any data been removed? How much? Which? Why?
- Who has access to this data?
- Are we collecting more data than we need?

Data science governance: The model perspective

Like data, models should have an owner (in agile terms often a product owner). Models should be subjected to automatic tests that cover both their overall performance (is the model better than baseline?), its specific performance (does the model handle some of the key special cases correctly?) and also that it generally works (does it fail correctly if it is given non-sensical input?).

Like data, model lineage is immensely helpful. What data was the model trained on? What settings were used during training? When was the model trained?

A model should be monitored. Both input and output. Over time, there is a risk that the world will change. For example, a model that uses wage levels as a parameter will need to be updated continuously with inflation.

Finally, targeted attempts to cheat and abuse the model should be monitored. This can often be done via input monitoring. Be aware of, for example, adversarial attacks (Goodfellow 2017).

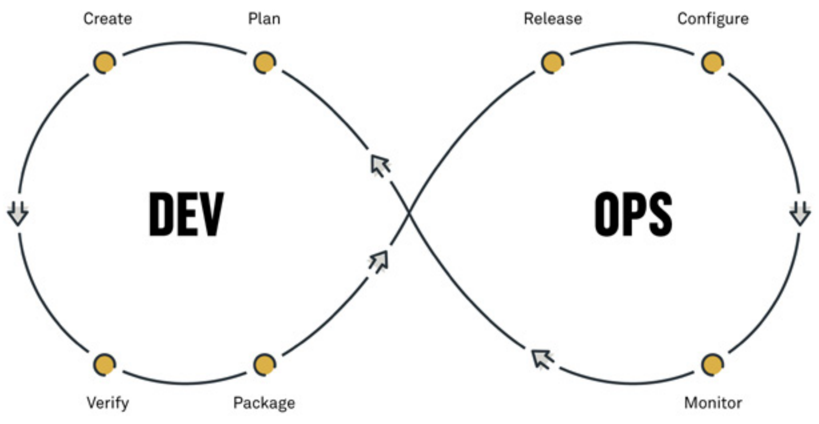
Key questions:

- Who owns the model?
- What are our expectations of the model's general performance?
- What are the most important edge cases that the model should handle? Does the model handle them correctly?
- How has the model been trained? On which data?
- How do we monitor input and input of the model? How big is our tolerance for changes in the data?
- How do we monitor and protect against cheating and abuse?

Data science governance: The code perspective

When developing data science, a natural part of the process is plainly to write a lot of code. Governance models in software development are a very mature field, and I recommend reading up on for example DevOps (Loukides 2012).

In a modern development environment, the data science solution should be included in a Continuous Integration/ Continuous Development (CI/CD) pipeline that includes unit testing of individual components along with a continuous integration test for the assembly of components from data entry to prediction. Tools such as GitLab, Jenkins or CircleCI make this process easy. For example, most of them can make a successful completion of automatic tests a prerequisite for automatic deployment (more on all these things in the next chapter).



A similarly important aspect of ensuring quality in data science is code reviews, where new additions or changes to a solution must be approved by another developer before being put into production. On small solutions, it may seem unnecessarily bureaucratic, but the value quickly becomes clear: the learning potential is great, and it captures and eliminates many mistakes.

Finally, code is a potential security risk. Cyberattacks start by identifying weaknesses in the code, and this can also be better protected, for example, through code reviews.

Key questions:

- Do we have a common development methodology (like DevOps, Scrum or SAFe)?
- What are our expectations of unit testing (both coverage and completion) of the individual components?
- Do we have a CI test? Is it automatic and a prerequisite for deployment?
- What is our practice for code reviews?
- How do we handle security issues in our code?

Summary

Deployment is the step where we begin to translate all of our data science into value, and governance is the system that ensures that we can make deployments easily and often.

Good governance ensures responsibility, processes, quality and security across data, models and code and does so with the least possible complexity, maximum consistency and with a glance to meet the strategic goals of the company.

If you can answer the questions below, you are well on your way.

	OWNERSHIP	PROCESSES	QUALITY	SECURITY
Data	Who owns the data, and who is responsible for it? Who is the data steward? What is our official definition of the data area?	Automatic reporting. Automatic logging of input and output of all major transformations. What data is being removed when and why?	Where does the data come from? What transformation has the data gone through? Is any data being removed? Does data look like it should at this point in the process?	Who has access to data? What data do we save? For how long? And where? And how is it saved?
Model	Who owns the model, and who has the responsibility for it?	How does the model perform in general? How does the model perform in specific edge cases? Does the model keep up performance over time?	Where does the model come from? What was the settings during training? What data was the model trained on?	Do we monitor the model for suspicious activity? Is the robustness of the model known? Are the weaknesses of the model equal for all users?
Code	Who owns this area of the code, and who has the responsibility for it? (Agile terms like product owner and scrum master can be helpful here).	Automatic unit testing of each individual function. CI/CD (Continuous Integration/Continuous Deployment) of the entire solution.	What is our practice for code review? Who is able to approve new deployments? Do we follow DevOps or a similar formal method?	Do we have a comprehensive overview of our risk? Do we have known security issues? Are they prioritised?

Chapter 8. Governance tools

In the previous chapter, I described relatively broadly the considerations one should make when starting to develop a governance model for data science. Governance is largely an organisational and human exercise that requires something as simple as meetings, appointments, trust, mandate and legitimacy.

But there are also a number of really helpful tools.

I will review a mere handful of my favourite tools, and many of them have great alternatives. I highly recommend anyone going into a data science project to consider these tools, but for small projects you can just select a few that make sense.

The tools:

1. Git
2. Metadata store
3. Logging
4. Automatic testing
5. Data profiling

Git

Git is the answer to all questions about review and traceability in code. Git is the most widely used version control system in the world. It is a way to have a central repository for all developers to access the projects' code and their specific version which they are working on. A repository is like a file drive, but a git repository has different branches. A branch is a version of the code base, and most often a developer will "branch out" to develop a new functionality to a system. A branch common to almost all git setups is the "master branch". This is usually the branch running in production and requires special rights to access. Commonly, there are also strict rules that the master branch must always work and requires passing certain automatic tests. In their own feature branches, developers can freely change, save (commit) and delete elements, and when they are done and a new feature is complete, they submit it for the master branch (create a pull request). Then, a series of automated tests will typically be run (more on that below), and a senior developer/data scientist will be asked to review the code. If the tests are passed, and the review is approved, corrections can be entered into the master version and "deployed" in the next round.

In short, the process looks like this:

1. Take out a new branch from master
2. Develop a new feature
3. Apply for a pull request (i.e. merge the new branch into master)
4. Testing
5. Code review
6. Approval and integration into master
7. Deployment

This is the rhythm which is often seen in well-functioning agile projects. In practice, it is oftentimes a lot more messy (and that's alright). Code review might lead to revisions that check out another branch, testing might fail or two different merges into master conflicts with each other and needs to be resolved. For this, I personally often use GitHub, but both Azure DevOps and GitLab are great alternatives.

Metadata store

A metadata store is a database that contains information on the models we have developed, and the data on which we base these models. The metadata store is the answer to questions like: Where does the model come from? What happened to data? How much data has been removed and why?

In the chapter on governance, I proposed to note down information about transformations of data and of the characteristics and performance of models on tests and training data (besides which test and training data they were trained on). But where? And how? Tensorflow Extended is probably the most mature answer, and Spotify's journey (Spotify_Labs 2019) gives some insights as to how.

If you do not want to depend on Tensorflow Extended, less can also do it. Any database, such as open source MySQL, can be used as a metadata store. Even a spreadsheet or table for models with an ID, a timestamp, the various parameters of the model and final performance on test and training data is a good place to start. One should then add a table for data transformations where there are columns with ID for the run, ID for the step of the run, name of the transformation, timestamp, how many data points came in, and how many data points came out of the transformation. Optionally, a step can be associated with a data profile that is stored on a disk separately (more on data profiles in a bit).

Logging

Logging (Ajitsaria 2018) is the process of monitoring and saving a log of a programme's actions. It is often a physical file on a disc, containing timestamps of when different actions were taken or potential errors or warnings were encountered along the way. Logging comes built into both Python and R, and there are also several third-party loggers (like loguru).

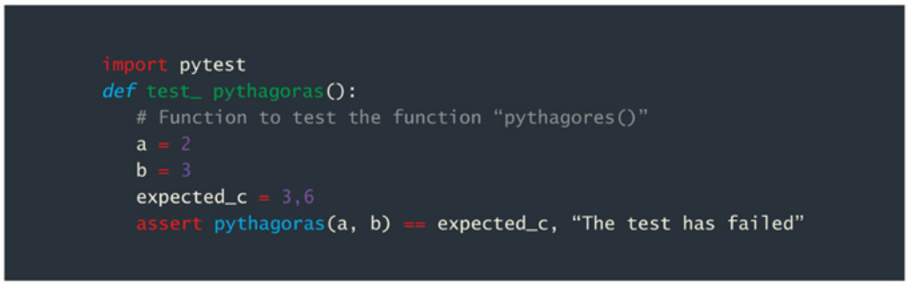
Logging example:

```
2020-04-16 16:35:11.997 | INFO | __main__:<module>:12 - Starting...

2020-04-16 16:35:11.997 | INFO | load_data:get_data_with_cache:13 - Loading from cache

2020-04-16 16:35:12.018 | INFO | load_data:get_data_with_cache:19 - Shape of dataframe: (17829, 6)

2020-04-16 16:35:12.051 | INFO | __main__:<module>:23 - Main input is: progress_to_plan

2020-04-16 16:35:12.076 | INFO | __main__:<module>:44 - Shape of X: (17472, 20) --- Shape of y: (17472, 1)

2020-04-16 16:35:12.159 | INFO | __main__:<module>:73 - All incomming variables: [...]

2020-04-16 16:35:12.162 | INFO | utils:__init__:41 - Constructing Data-object:
```

Automatic testing

Tests are about detecting errors before they occur. Automatic tests have the benefit that they can be used frequently and at no cost. Generally, there are two types of tests: unit tests that test a specific functionality, and a continuous integration test that tests a system from start to finish.

Unit tests are built into Python and R and are also supported by most development tools (such as PyCharm and VS Code) in addition to various Git tools (such as GitHub and GitLab). They are small pieces of code that could look like this:



```
import pytest
def test_ pythagoras():
    # Function to test the function “pythagores()”
    a = 2
    b = 3
    expected_c = 3,6
    assert pythagoras(a, b) == expected_c, “The test has failed”
```

Ideally, all features developed and brought into production should have unit tests. Unit tests ensure that if other developers change parts of the code, the features will continue to produce the expected results.

Unit tests can also support documentation and provide examples of expected input and output of functions for new developers.

Unit tests live in their own test files and can be set up to generate automatic reports. They can be set up to be prerequisites for creating pull requests in Git and act as a form of automatic governance.

I am a big fan of PyTest (Krekel 2020), but there are other good tools like Robot and the built-in unit test.

Continuous integration tests (CI tests) are larger tests ensuring that a data science solution works from start to finish. Typically, a CI test has access to a set of sample data and runs through all transformations one by one, including final drawing predictions from the final model. The CI test should be seen as a form of automatic general examination before deployment and ensure that all the individual parts work together as a whole. A CI test is not focused on the quality of the output of the model (this should be done during the development of the model in the test-train phase) but solely on the question: Does the model actually function?

Data profiling

Data profiles are standard reports of a dataset. The number of variables and observations. The number of missing data points and different properties of the individual variables. The trick is to automatically create these data profiles either through a tool, such as Pandas Profiling, to throw data files that can be visualised into a PowerBI dashboard or through a tool like Google Facets. (See the chapter on data visualisations for elaboration on these three tools.)

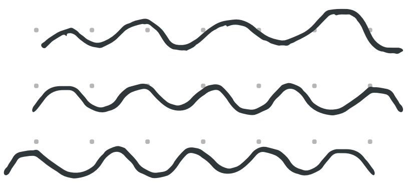
Data profiles can be costly to calculate for large datasets and should therefore be used more sparingly. For example, it can be at the beginning and end of a complete data flow or in connection with the most important data transformations. Data profiles can advantageously be linked to metadata via IDs. A lot of data profiling tools also allow saving as either PDF or HTML, which means that they are portable and easy to share among data scientists but also business functions, if necessary.

Data science for you

I hope that this guide has provided you with an overview of the data science process – not just from data to model but from problem to solution. As I wrote in the beginning, data science is a combined discipline of both software development, machine learning and statistics and business/domain knowledge. None of the three parts must be forgotten. I hope that this guide can serve as a starting point for your own data science journey or to help others create value for organisations through clever use of data.

Finally, I have included a list of references to various resources mentioned in the guide. These are both articles and books, but perhaps more importantly, a comprehensive list of all the software tools that have been mentioned.

Should you have any thoughts, questions, wishes, corrections or anything else, do not hesitate to reach out! I would love to hear from you and help wherever I can.



References

Wikipedia. 2020. Wikipedia: Data Science. 21 April. https://en.wikipedia.org/wiki/Data_science.

Corti.ai. 2020. Corti.ai. https://www.corti.ai/.

Hede, Adam. 2020. Implement Learning Institute, Data Science Master Class. April. https://learninginstitute.implement.dk/courses/data-science-masterclass.

scikit-learn. 2020. scikit-learn.org/stable. https://scikit-learn.org/stable/.

W3Schools. 2020. SQL LEFT JOIN keyword. https://www.w3schools.com/sql/sql_join_left.asp.

Grolemund, Garrett, og Hadley Wickham. 2020. R for Data Science. https://r4ds.had.co.nz/tidy-data.html.

Ascombe, F. J. 1973. »Graphs in Statistical Analysis.« The American Statistician 27:1, 17-21.

Matejka, Justin, og George Fitzmaurice. 2020. »Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing.« Autodesk Research.

Google. 2020. Facets. https://pair-code.github.io/facets/.

Wattenberg, Martin, Fernanda Viegas, og Johnson Ian. 2016. »How to use t-SNE effectively.« Distill.

Microsoft. 2020. PowerBI. https://powerbi.microsoft.com/.

Tensorflow, Google. 2020. Google Tensorflow. https://projector.tensorflow.org/.

Profilling, Pandas. 2020. Pandas profilling. https://github.com/pandas-profiling/pandas-profiling.

PAIR_Research. 2020. People + AI Guidebook. https://pair.withgoogle.com/guidebook/.

ICO. 2020. Information Commissioners Office. https://ico.org.uk/.

Wikipedia. 2020. Differential Privacy. https://en.wikipedia.org/wiki/Differential_privacy.

Muhajlovic, Illuja. 2020. Towards Data Science. https://towardsdatascience.com/how-artificial-intelligence-is-impacting-our-everyday-lives-eae3b63379e1.

Seeing Potential, Google. 2020. Seeing Potential. https://about.google/stories/seeingpotential/.

Enversion. 2020. KAUNT. http://enversion.dk/okonomi/.

Vincent, James. 2019. The Verge. https://www.theverge.com/2019/6/27/18760896/deep-fake-nude-ai-app-women-deep-nude-non-consensual-pornography.

Enversion. 2020. Sundhed. http://enversion.dk/sundhed/.

O'Neil, Cathy. 2016. Weapons of Math Destruction. Crown Books.

Synced. 2019. Humans Don't Realize How Biased They Are Until AI Reproduces the Same Bias, Says UNESCO AI Chair. https://medium.com/syncedreview/humans-dont-realize-how-biased-they-are-until-ai-reproduces-the-same-bias-says-unesco-ai-chair-9968bb1f5da8.

Knight, Michelle. 2017. Dataversity: What is Data Governance? https://www.dataversity.net/what-is-data-governance/.

Goodfellow, Ian. 2017. Adversarial Example Research. https://openai.com/blog/adversarial-example-research/.

Loukides, Mike. 2012. O'Reilly, What is DevOps? http://radar.oreilly.com/2012/06/what-is-devops.html.

Git. 2020. Git. https://git-scm.com/.

GitLab. 2020. GitLab. https://about.gitlab.com/.

Jenkins. 2020. Jenkins. https://jenkins.io/.

CircleCI. 2020. CircleCI. https://circleci.com/.

GitHub. 2020. GitHub. https://github.com/.

Azure. 2020. Azure DevOps. https://azure.microsoft.com/da-dk/services/devops/.

TensorFlow. 2020. TensorFlow Extended. https://www.tensorflow.org/tfx.

Spotify_Labs. 2019. The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow. https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/.

MySQL. 2020. MySQL. https://www.mysql.com/.

Python. 2020. Python.org. https://www.python.org/.

Project, R. 2020. The R Project for Statistical Computing. https://www.r-project.org/.

Ajitsaria, Abhinav. 2018. Logging in Python. https://realpython.com/python-logging/.

Delgan. 2020. Loguru. https://github.com/Delgan/loguru.

JetBrain. 2020. PyCharm. https://www.jetbrains.com/pycharm/.

Microsoft. 2020. Visual Studio Code. https://code.visualstudio.com/.

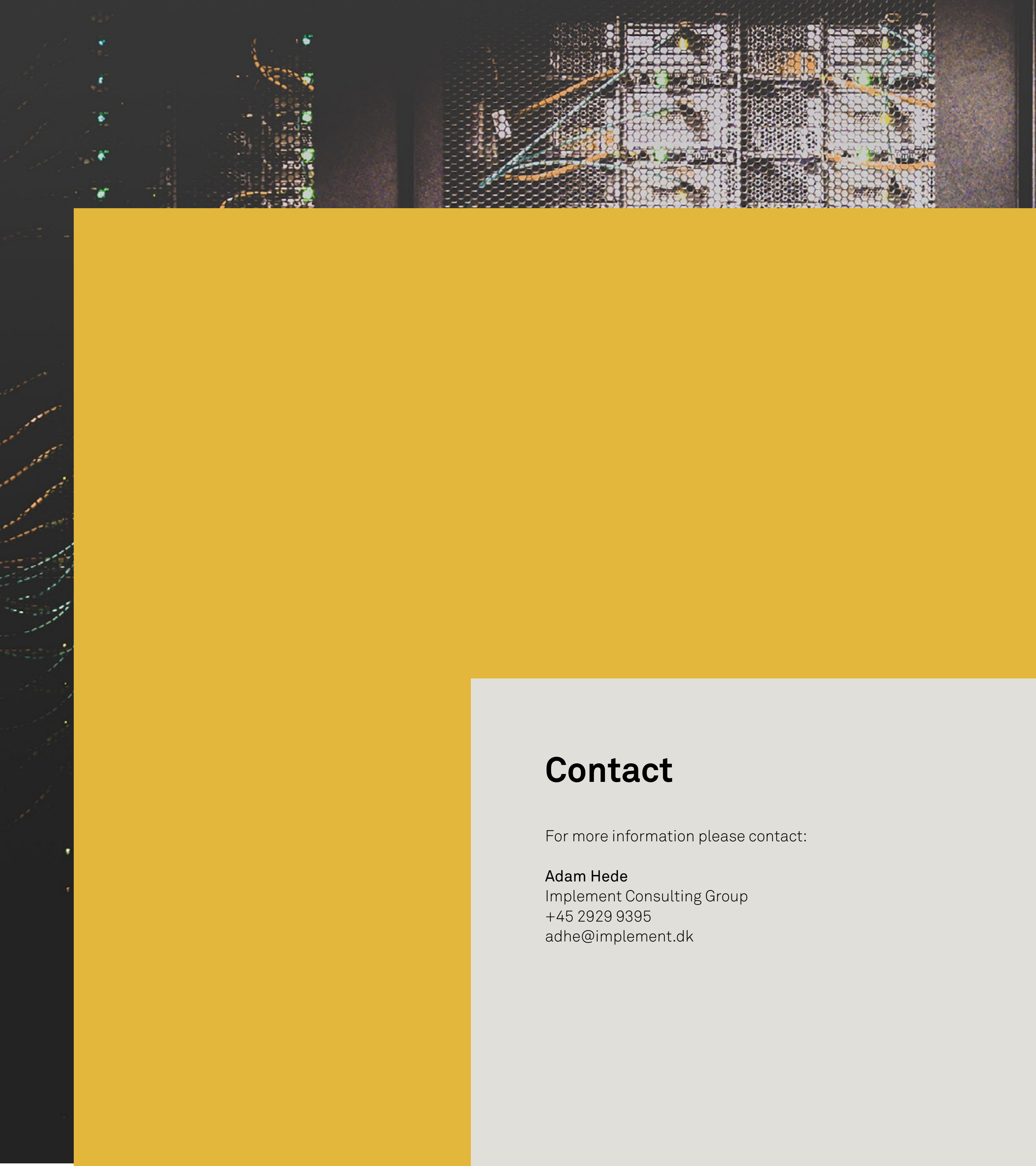
Krekel, Holger. 2020. PyTest. https://docs.pytest.org/en/latest/.

Robot. 2020. Robot. https://robotframework.org/.

Foundation, Anaconda. 2020. Anaconda. https://www.anaconda.com/.

Pandas. 2020. Pandas. https://pandas.pydata.org/.

Matplotlib. 2020. Matplotlib. https://matplotlib.org/.



Contact

For more information please contact:

Adam Hede
Implement Consulting Group
+45 2929 9395
adhe@implement.dk