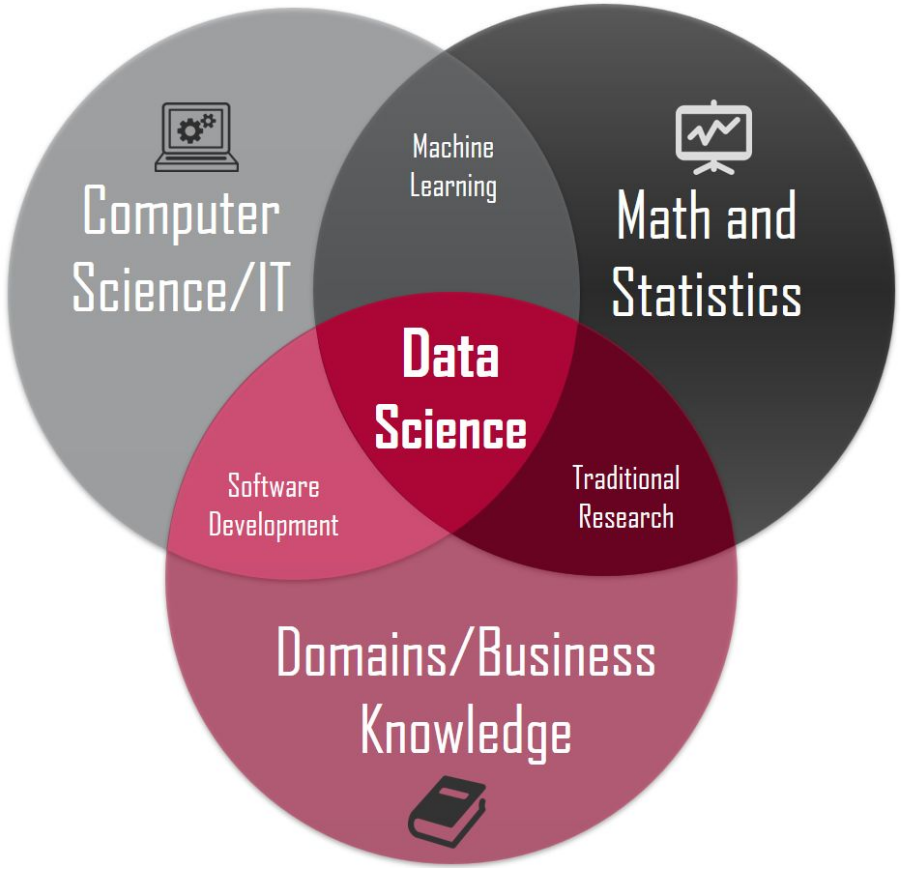
Deploying Machine Learning Models

Incorporating Software Development Practises and Reproducibility in Data Science

Carl Stanley

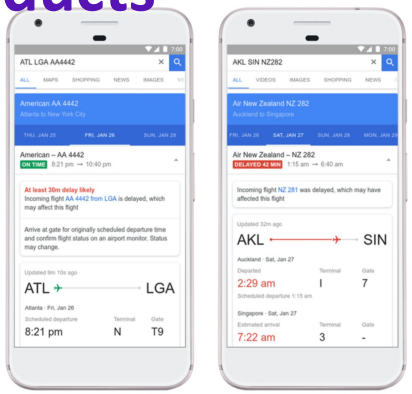
Data Science is one of the most popular professions in the last few years, often being called a “hyped” job. It’s especially popular in my flat (hi Jake).

- Data science is a pretty new field in IT
- Comprises of solving business problems with code – however you want to tackle it
- A few different areas that can be specialised in:
  - NLP
  - Deep Learning
  - Etc.

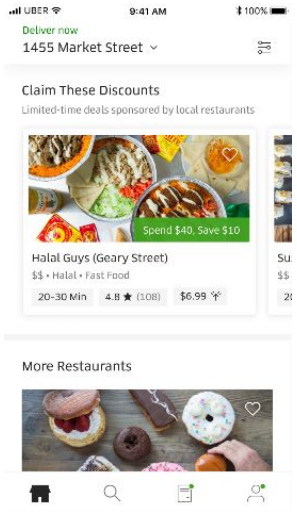


Data Science business use cases range from fraud detection and cyber-security, to incident prevention and recommending products

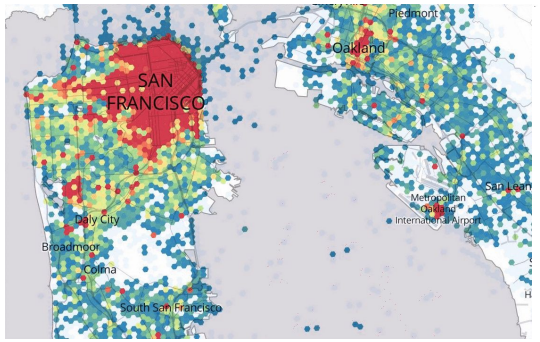
Vodafone & Telefonica(O2) "**Cognitive** **lligence**"



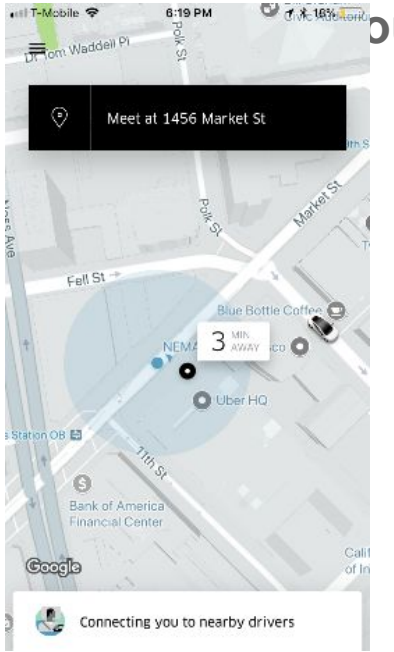
Google Flights uses Machine Learning to warn users about delays, and even **predict them before they're** **ounced**



UberEats uses real time Machine Learning models to **suggest** restaurants and menu items



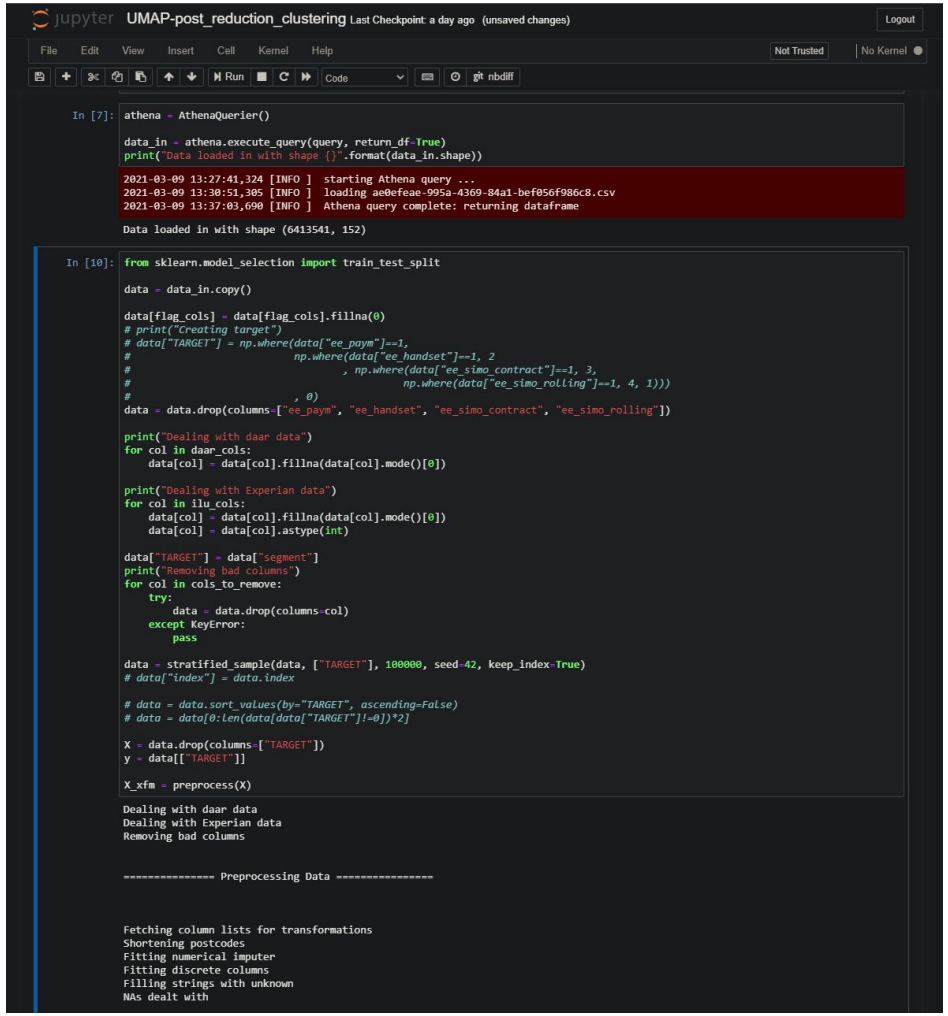
Uber use Machine Learning to **predict** where rider **demand** and driver-partner **availability** will be at various places and times in the future.



Uber use a Machine Learning model to **predict** ETA routing **errors** and then use the predicted error to make a correction, to **improve accuracy** and **user**

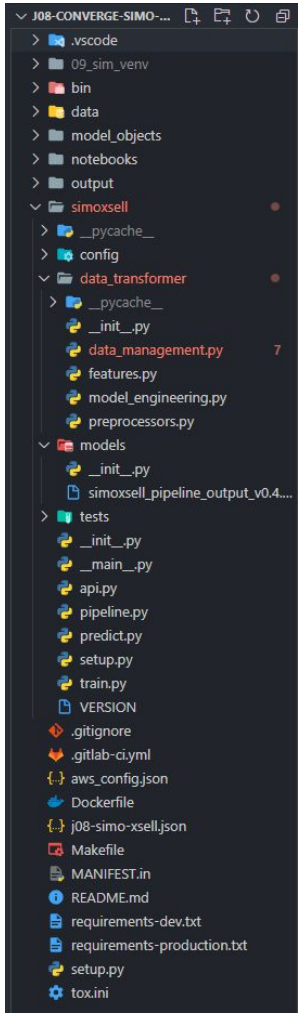
Experimentation is Key in Data Science – the more curious you are, the better your results.

- Jupyter notebooks give you a really easy way to interact with data
- Quick plots, analysis, and markdown integrated
- Can export it as HTML to chat through



When you’re done with experimentation, you need to make your code reproducible by packaging your work

- Once ready to use a model at scale, you need to create a python package for it
- This is not optional (most of the time)!
- You can automate a lot with CI/CD but the end product should be a callable package to run code against your model

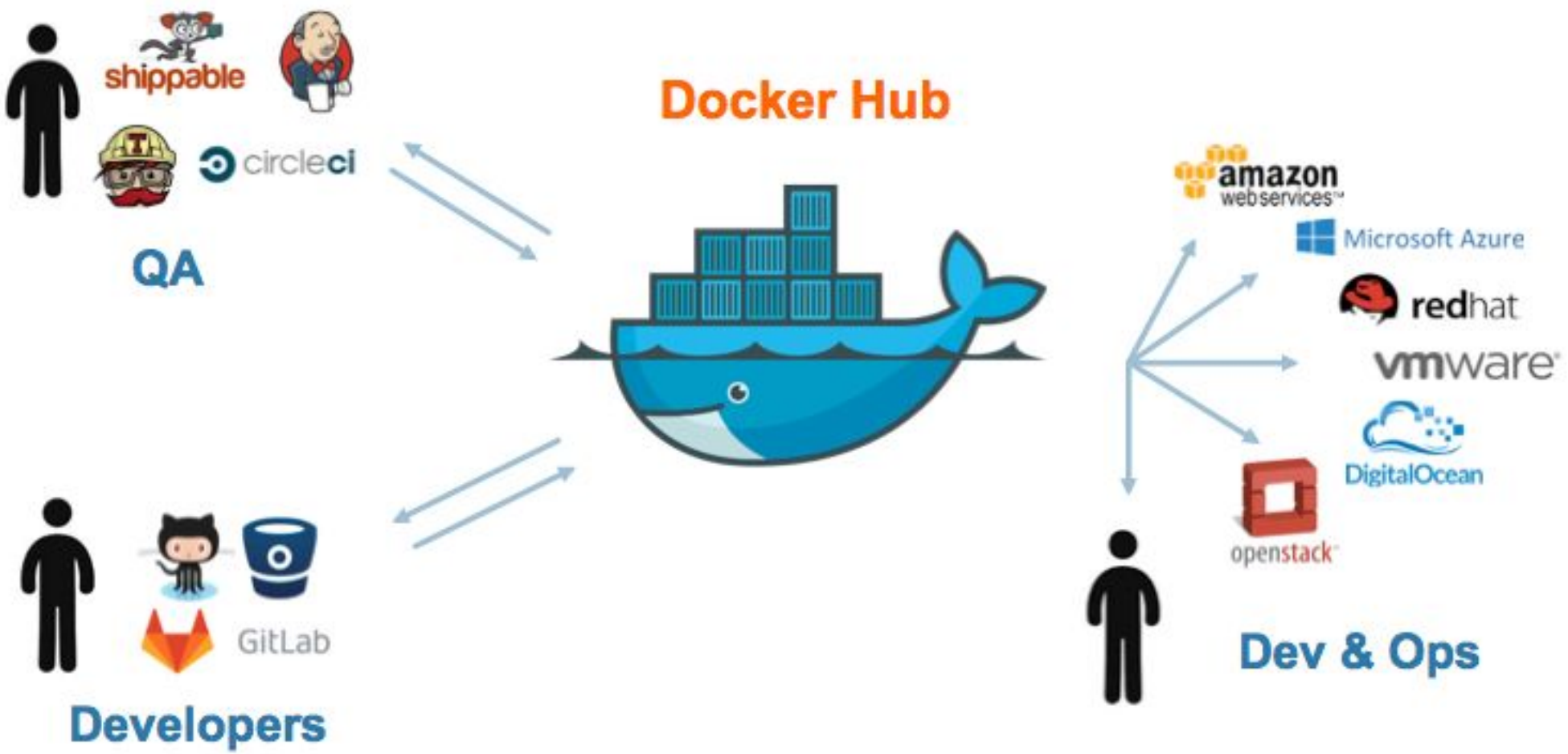


Docker is industry standard for ensuring developers utilise the same exact environment, no matter where they are

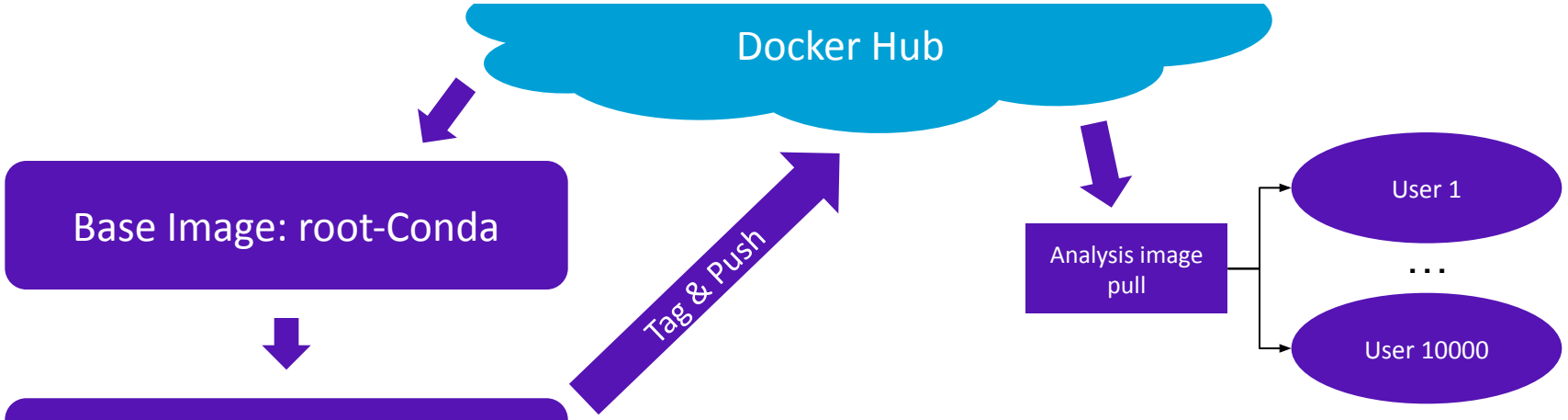
- Docker enables developers to produce a reproducible environment through a concept called “containers”.
- Traditionally, you set up your env, run your code, then run tests; Docker does all of this at once
- Docker provides a standard framework to ship / deploy / scale your code

What does this mean?
Building 1 docker image for a use case (i.e. ROOT Conda) ensures identical set-up each time, reducing issues with tertiary versioning etc., then overlay your model code and push!

Docker Hub is the source of all of truth for your images – it's GitHub for containers



An Aside – Docker isn't just useful for data science! It can be used to speed up development on any analysis




```
FROM rootproject/root-conda:latest

RUN git pull <my_cool_project>@master
RUN pip install –r requirements.txt

# Stuff to run analysis that takes 1000 lines
# of code for some reason

ENV IM_COOL="yes"

ENTRYPOINT["/usr/src/", "entrypoint.sh"]
```

Outcome

A reusable, scalable version of your analysis that anyone & everyone who owns Docker can use.

Benefits

- No more environment issues
- Code is shared automatically when pushed to docker hub
- Gitlab has its own docker hub!!

Jupyter Notebook > .py > Docker Example

Notebook (.ipynb)

How Code is Run
- In Cells
- Allows for rapid experimentation
- Works quickly and in *modular fashion*

Who can Run the Code
- Person with the notebook with GUI

How Tests are Done
- They aren't

Results
- Quick prototyping
- Interactive shell
- Visual outputs for quick analysis

Package (.py)

How Code is Run
- From terminal
- Functional / OOP
- Runs as a package (if you're good at writing code)

Who can Run the Code
- Anyone with the same env as you
- Projects usually contain a requirements file and a README (not always…)

How Tests are Done
- Manually with pytest (if you've written them)

Results
- End to end (e2e) process, run all at once
- Open ended regarding deployment

Dockerfile

How Code is Run
- From terminal, inside container
- Code often cloned straight from Git

Who can Run the Code
- Anyone with docker

How Tests are Done
- Tests are already done when the container is made
- Can add more tests with pytest if you want

Results
- Shippable product that anyone can use with 1 line

CI can enable trust in your work with less active effort, and CD enables agile development and lets developers learn quickly

Continuous Integration

- Automated Integration Testing
- Dynamic environment building
- Ensuring **reproducible outcomes**
- **Differential Testing** of ML Models
- Automated simple unit testing
- Code linting

Continuous Deployment

- Automatic environment creation & updating
- Integrated **environment segregation** (dev, test, pre-prod, prod) based on Git branches
- Scheduled activities

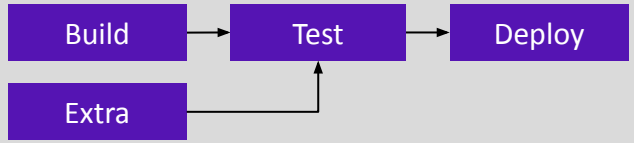
BT

Data Science isn't just a modelling career; you can't claim you've made a meaningful model until you're actually helping a business achieve it's objectives

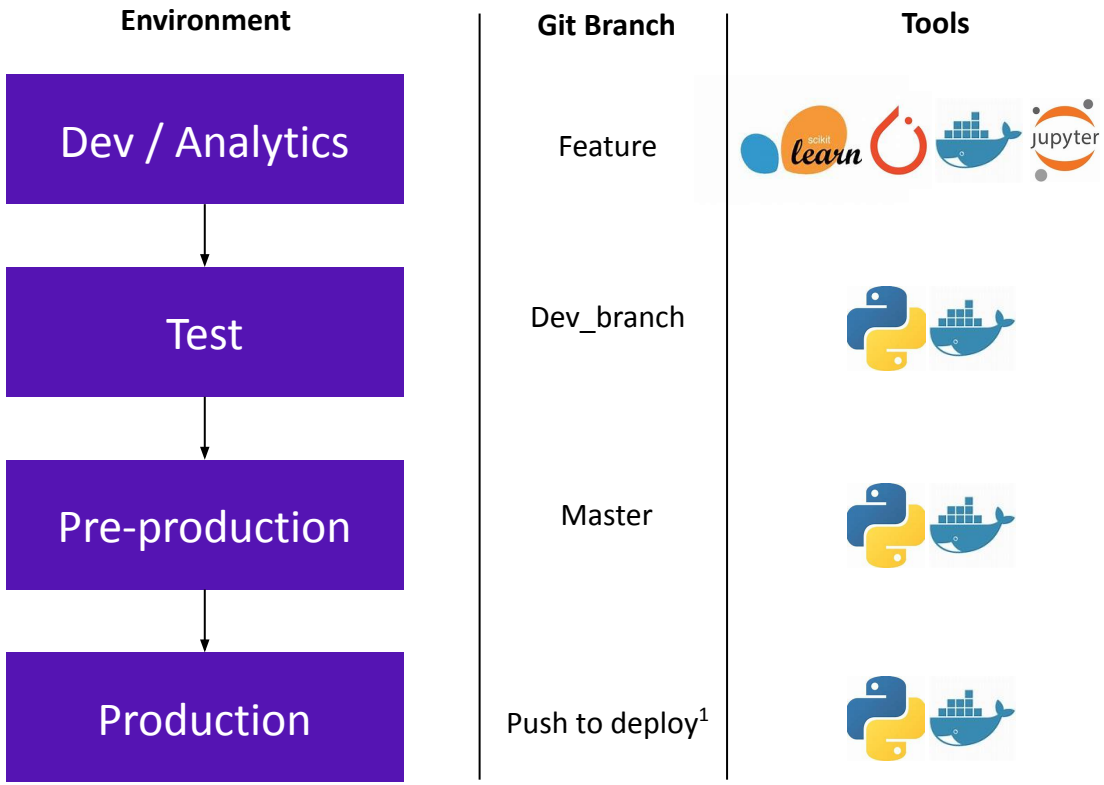
Environments

- **Development** is an experimental area where data scientists have data and a platform to utilise it (what we have today)
- **Test** is a place to test integrations and deployments with no outside exposure
- **Pre-prod** is a production clone for agile deployment and stakeholder engagement

CI Workflow



- Build: Builds code and ensures all versioning etc. works (it starts)
- Test: All unit tests, integration tests, and differential tests are run (it does what you expect)
- Deploy: Expose the code to an environment for consumption (it's ready!)



Version control + containers + CI/CD together form a great foundation for agile, reproducible data science

1. Maybe

Example – Using ML to reduce noisy data in an experiment

Step 1 – Understand the business problem

Step 2 – Gather data

Step 3 – Experiment with feature engineering & Preprocessing

Step 4 – Modelling

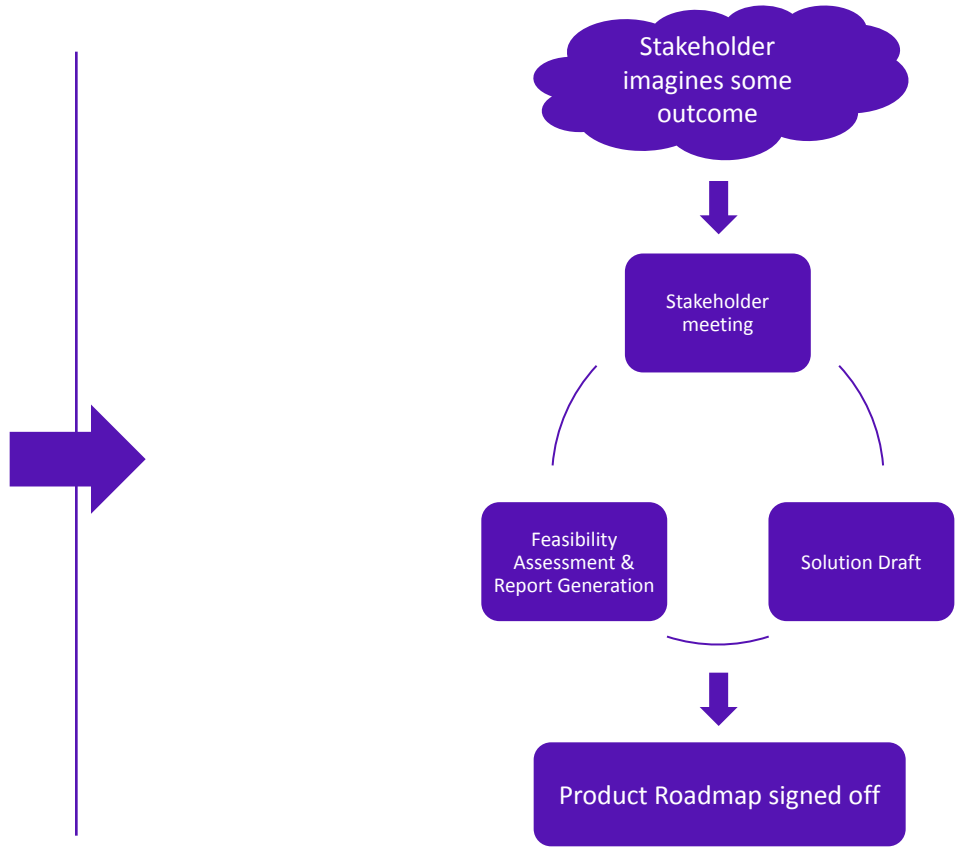
Step 5 (where most people stop) – Package the project for quick retraining

Step 6 – Containerise and deploy model to serve at scale

Step 7 – Monitor performance

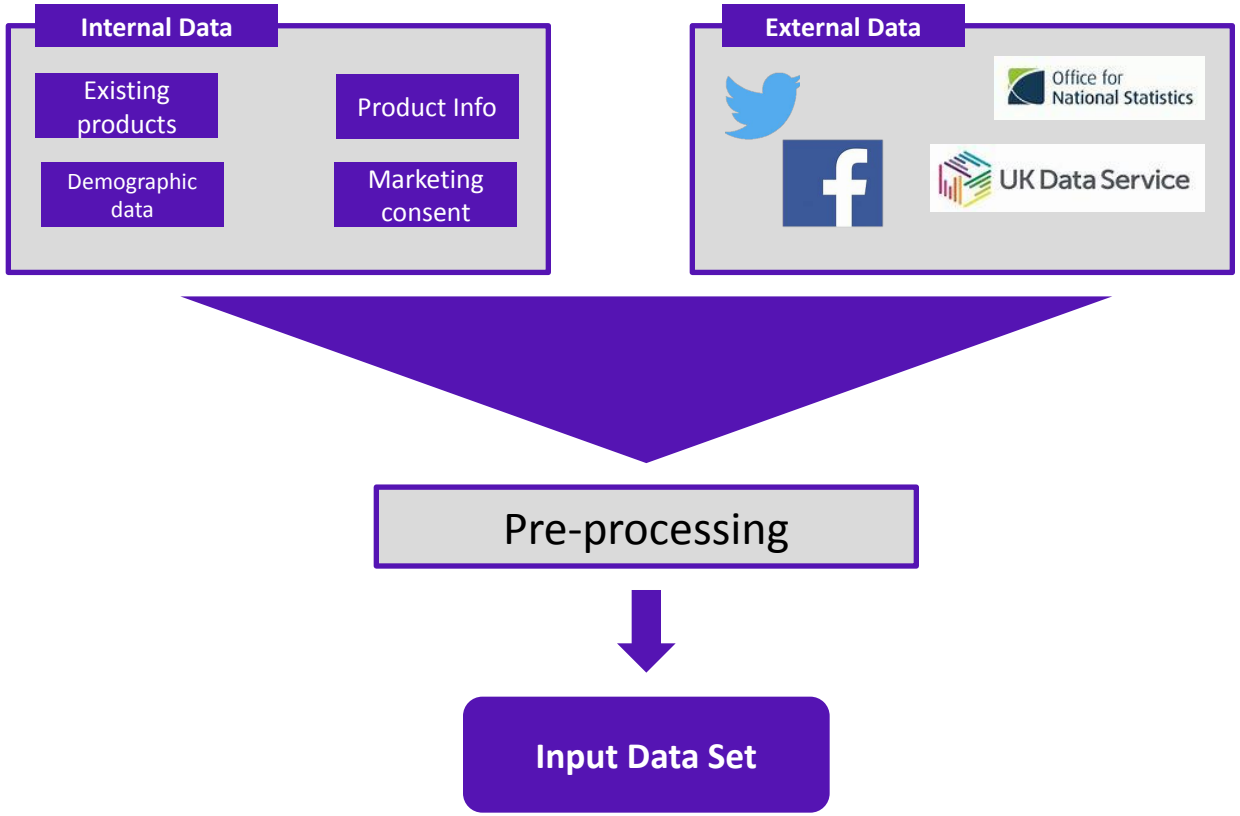
Step 1: Understanding the business problem is (obviously) key, but most of the time the stakeholders won't know what they want when they ask for it

- Stakeholders have a *very* different perspective on business problems
- Solution design is a process – don't rush it
- You're selling your capabilities



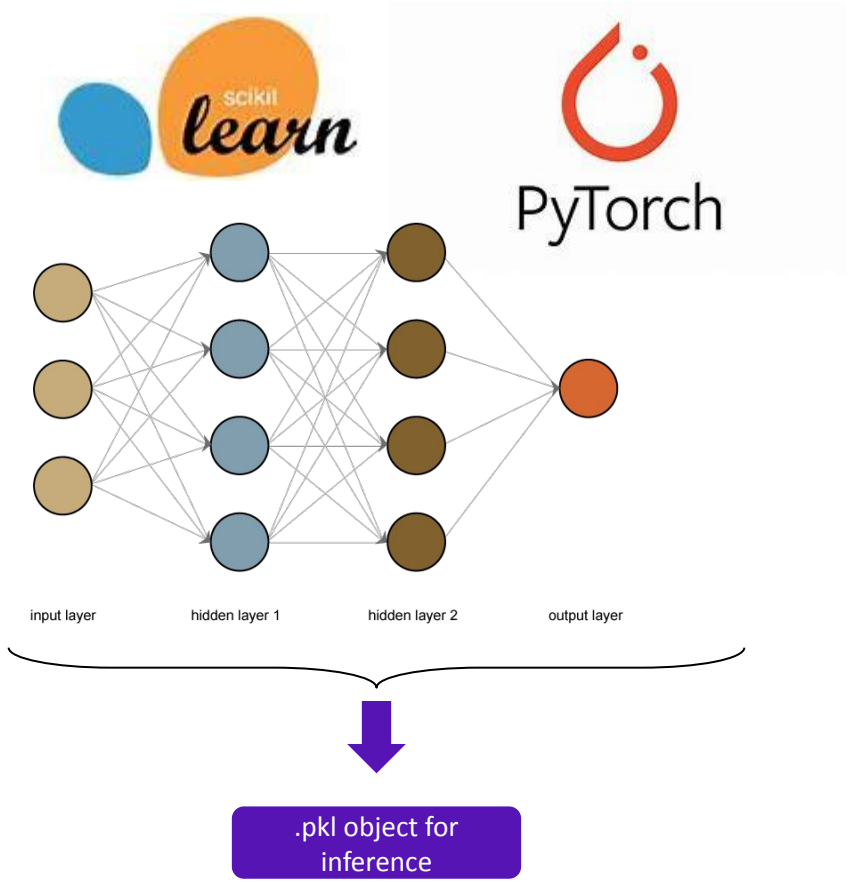
Step 2: Gather data – This is where the most value can be gained from a model, so get creative; you're trying to best describe your base for the problem

- This is where the bulk of the work takes place
- Creating your dataset can take a long time and feel like you're not making progress
- Skipping step 3 as it's very case-by-case sensitive



Step 4: Modelling – Unfortunately the simplest solution is *always* the best one, especially in a business context

- This is normally the easiest part of the job – especially if you've done a good job in step 2/3
- Keep in mind what you're trying to achieve from a business perspective
- Keep asking yourself 2 questions:
  - How is this improving on the current business process
  - How is this going to be consumed



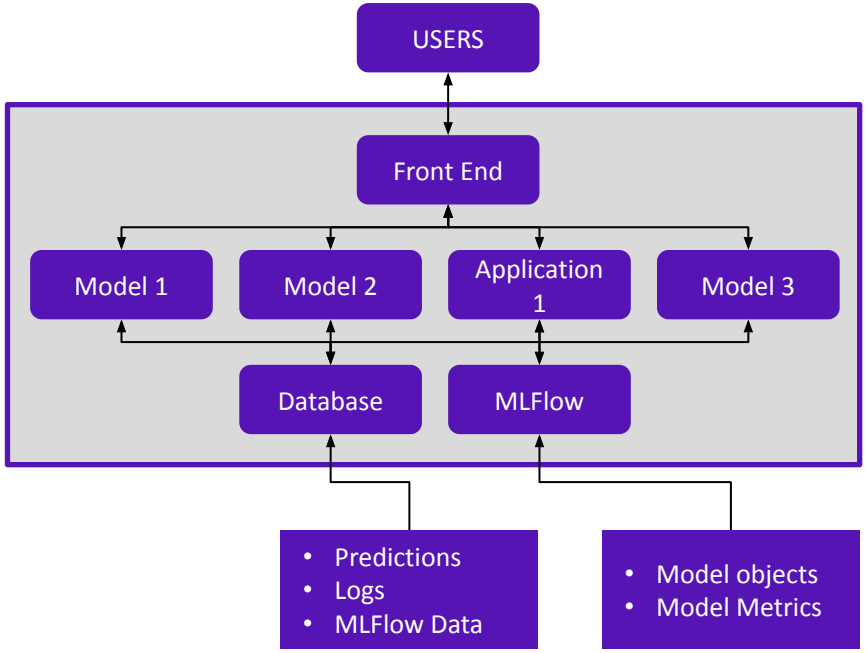
Step 5: Packaging – If you've built a big fancy model that takes 6 hours to provide inference, you're going to have a bad time

- This is where you enable your model to be called in 1 line of code
- It opens up a world of possibility for how it can be consumed
- Packaging your model and wrapping it in a Flask server enables live predictions at scale

```
"""Frontend for flask app."""
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

import pandas as pd

from simoxsell.predict import make_prediction, run_batch_scoring
from simoxsell.train import run_training
from simoxsell import __version__ as _version

from aws_tools.s3_tools import send_scoring_output
from aws_tools.cloudwatch_logging import logger as _logger


server = Flask(__name__)


@server.route("/simoxsell/helloworld")
def helloworld():
    """Return Hello World for API Testing."""
    _logger.debug("Successfully called route.")
    return "Hello World! You're on the API for simoxsell!"


@server.route("/simoxsell/make_prediction", methods=["GET"])
def make_test_pred_from_flat_file():
    """Make a test prediction to demonstrate model."""
    _logger.debug("Successfully called route.")
    _logger.info("Making test prediction from Athena...")
    data = pd.read_csv("/app/data/cs_test_data.csv")
    preds = make_prediction(data)
    _logger.info(preds)
```

Step 6: Containerise & Deploy – Most deployment options opt for Docker Containers anyway, so you're a step ahead of the game

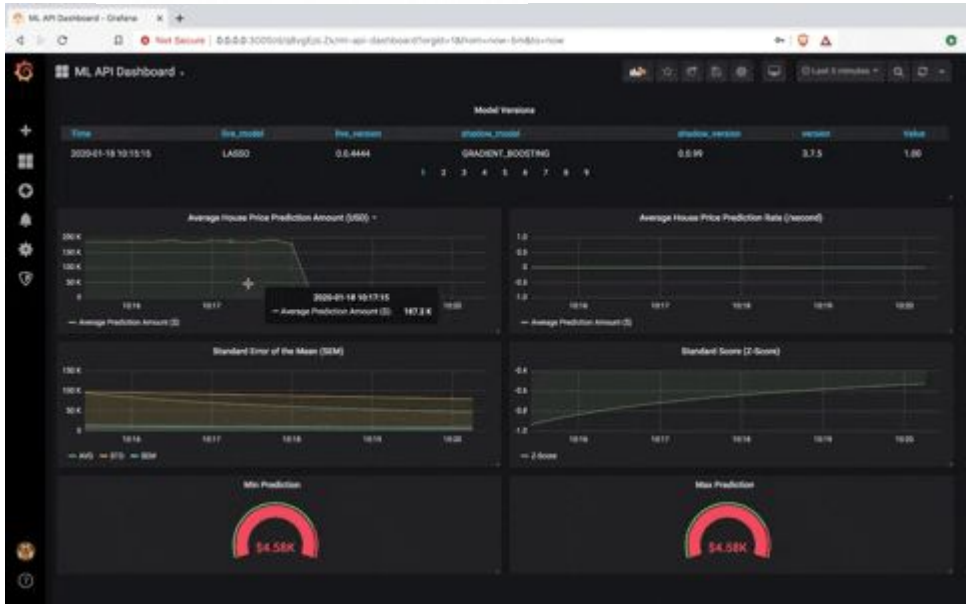
- Unfortunately, most businesses disable 90% of Cloud Infrastructure features
- There is a grey area between data science & data engineering here – the ML Engineer is here to fill the boots
- Microservices are cool and hip right now



Stage 7: Monitoring – This is a very poorly done step in general; if this is done right it really helps the lazy developer move on with their life

Why is Monitoring not done as much?

- It's boring
- Businesses aren't mature enough to challenge data science effectively
- It takes a lot of effort to do right



What should we be monitoring?

- Output metrics: The prediction results as & when they're made to check they're reasonable
- Online performance: A comparison of the prediction to the real result, so that we can understand real performance
- Machine-oriented metrics: API response times, CPU usage, etc.

Thanks for listening!

Questions?

Fig 1. Marcella's EPP class, 2016, colourised