Planning and executing Machine Learning project

2017-11-04

Few examples

- What is most dominant element in the image
- What is the sentiment of the text - negative / positive
- Recommend article/product to customer
- Categorization of text into one of "buckets" - e.g. urgent message, low priority, spam
- Product rating prediction
- Product purchase prediction
- Chatbots
- Language translation
- Speech recognition
- Sales prediction
- Stock price prediction
- What is the temperature going to be tomorrow?

What is machine learning?

A Tool !

Task performed by human

Task performed by machine

Presentation plan

Project stages:

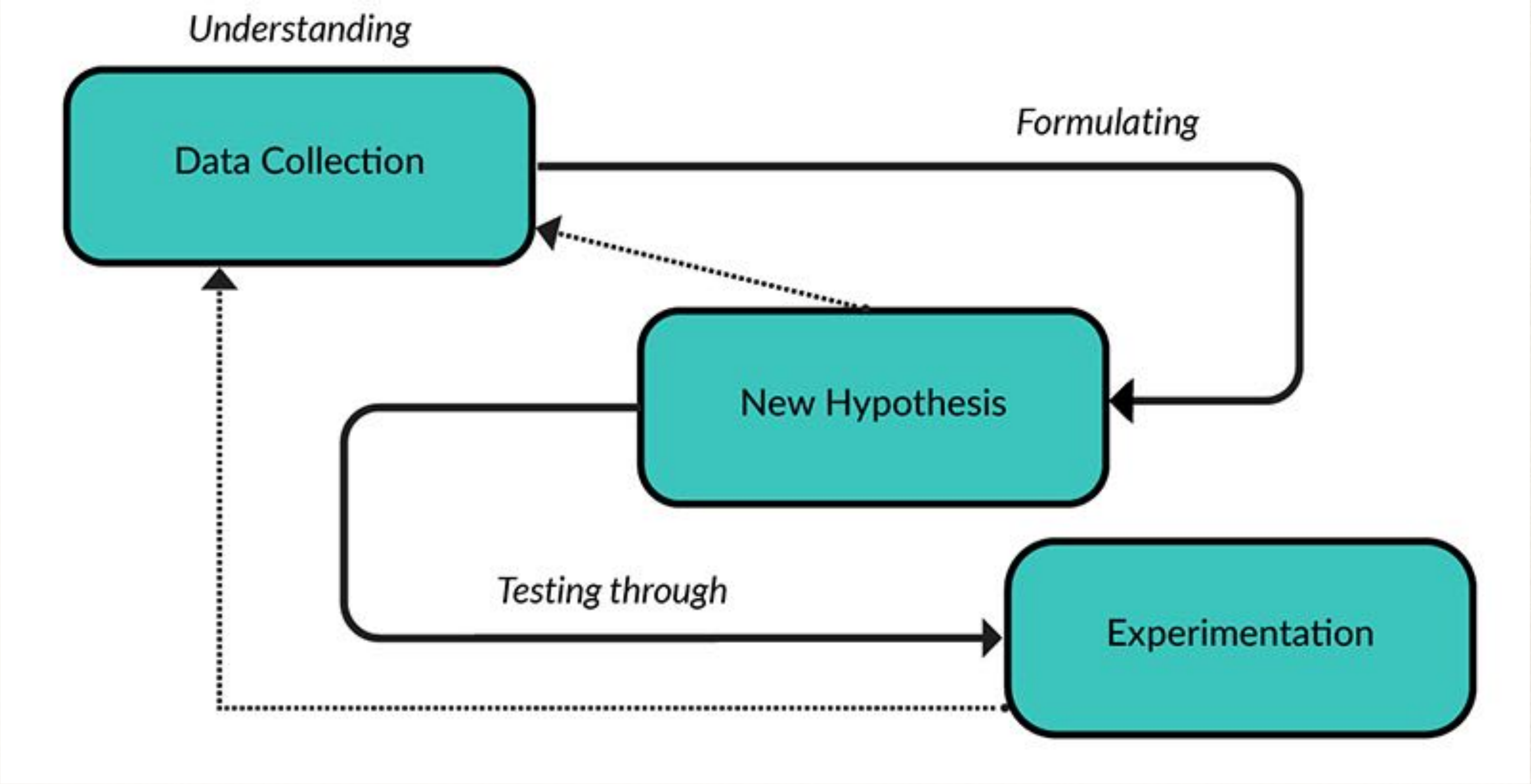
- Planning
- Executing
- After release

Planning

Software engineering vs. data science

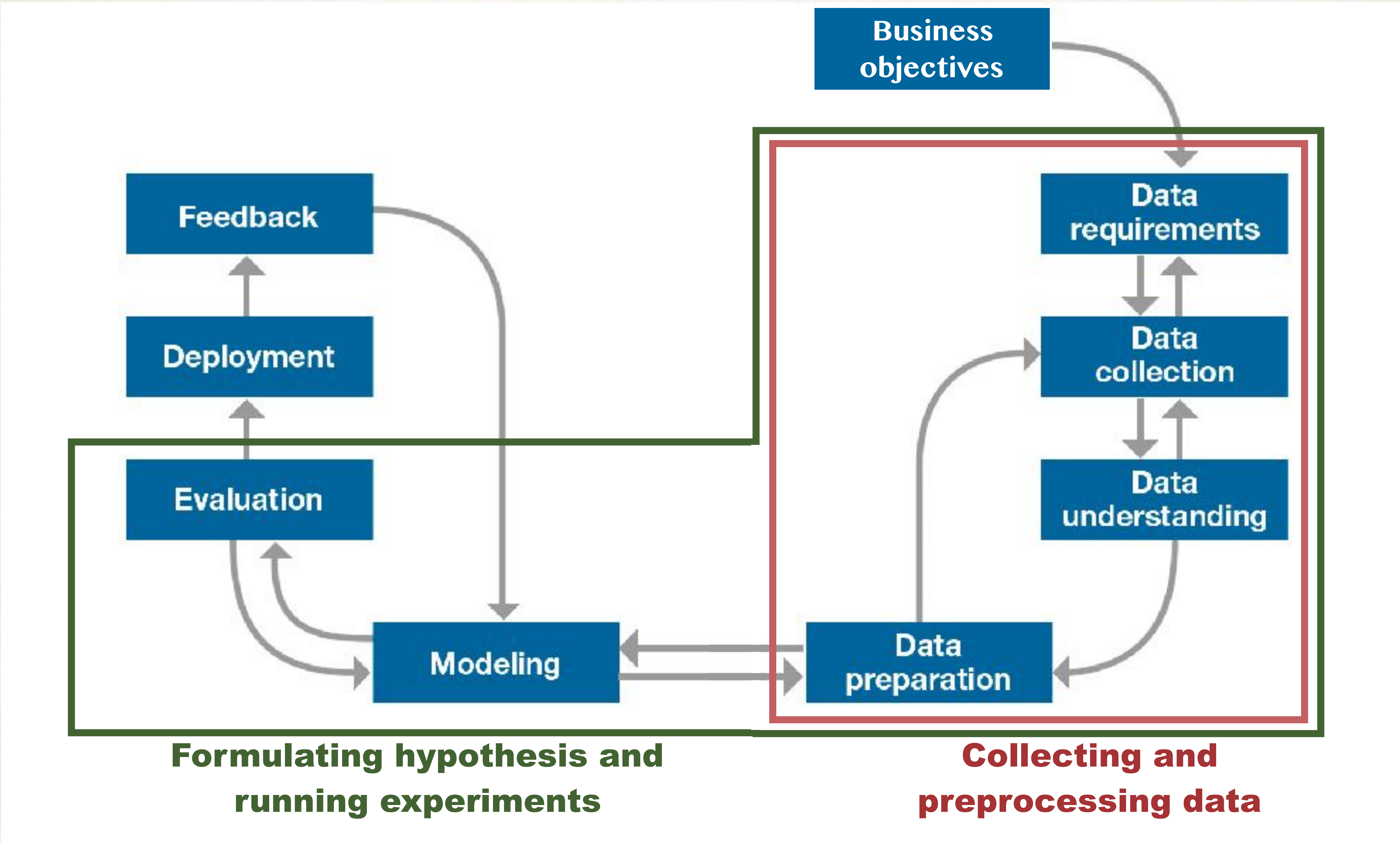
	Software	**Data science**
Project goal	build product features	build best performing model
Long term process	keep adding/ improving features	keep improving model to fit current business needs

Scientific method in data science



source: https://www.thedatalab.com/about-us/what-is-data-sciene

Data science - process



Choose your (model's) goal

- Metrics:
  - Accuracy (% of examples correct)
  - Precision (% of detections that are right)
  - Amount of error (for regression problems)

- When your prediction is good enough for production?
  e.g. for product recommendation - 70% accuracy might be enough

- Can you <u>hide</u> results when you're not certain about your prediction?

How difficult is my problem?

- Who else did something similar?
- Google for ready solution in articles, blogs, scientific papers

Build, find ready model or buy

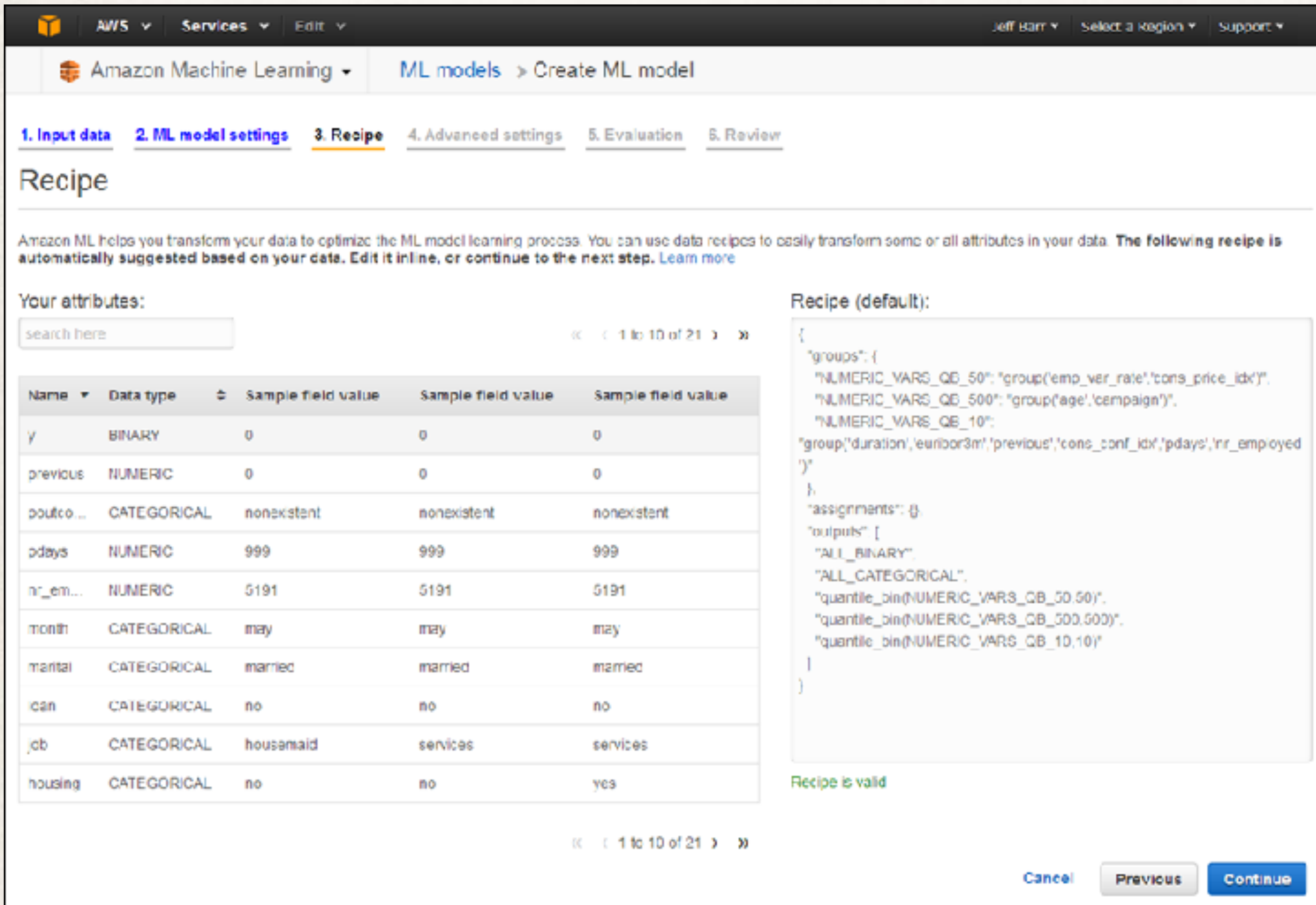
Build your model from scratch

Find ready service

If not available or not good enough, build from scratch

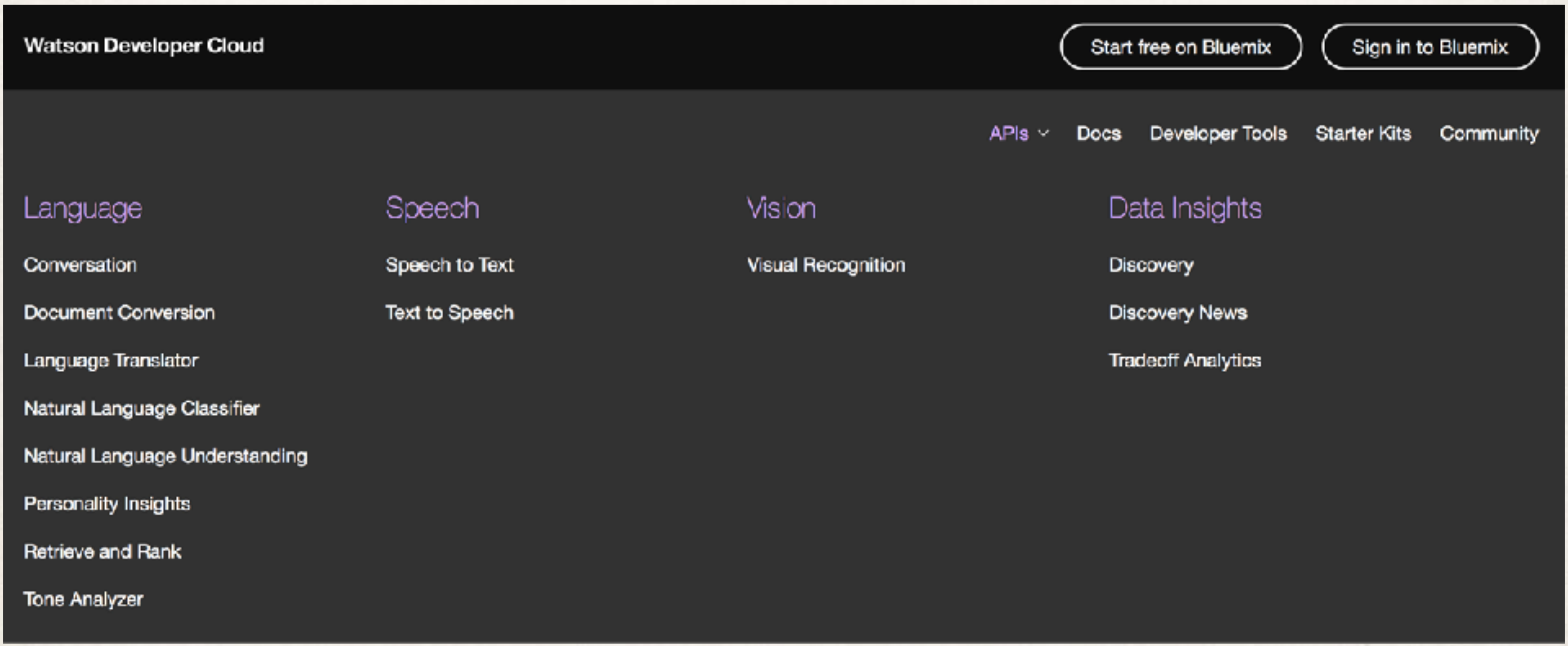
Ready services

Amazon Machine Learning



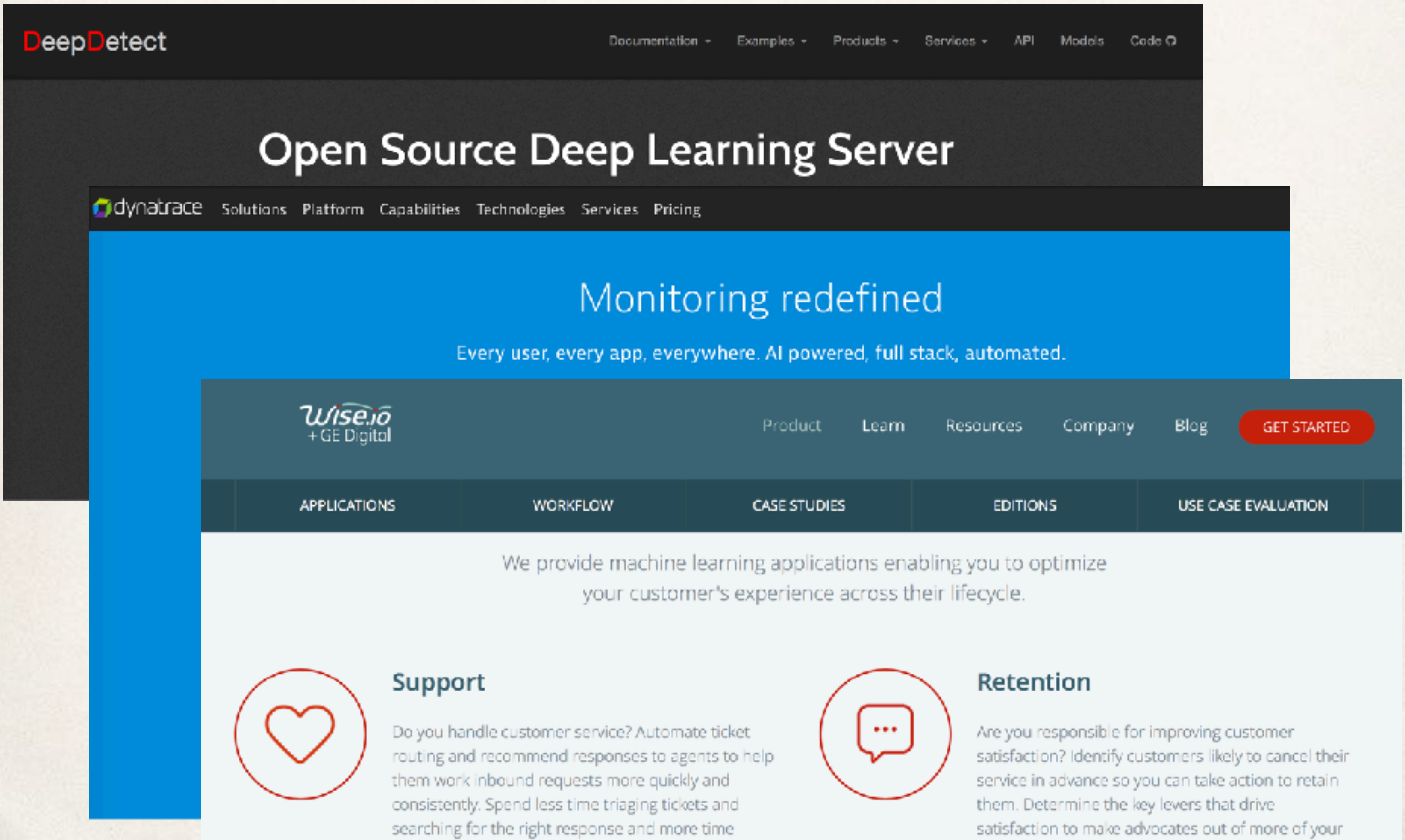
Ready services

Watson Developer Cloud



Ready services

Other SaaS



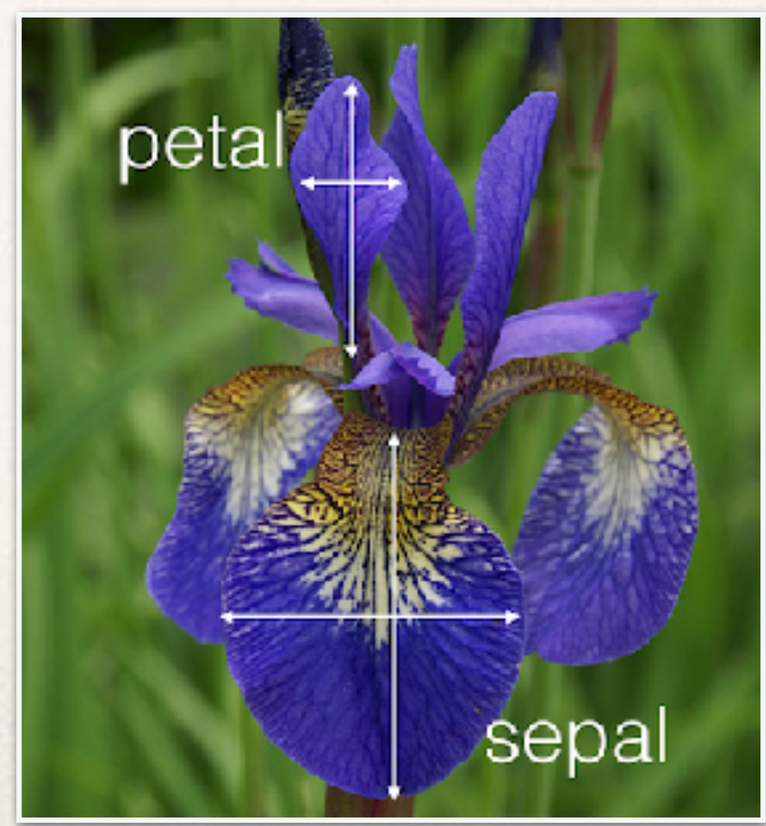
Ready service doesn’t solve all problems

Buy	Build	
✓	✓	Collecting and labeling data
✓	✓	Data cleaning and preparation
	✓	Building and training the model
	✓	Building separate application to host the model
✓	✓	Building API to communicate with your model
	✓	Preparing production environment and “go live”
	✓	Maintenance

What data will I need to build my model?

Of course depends on problem, but minimum requirement is:
What data would human need to solve this problem?

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*
5.4	3.7	1.5	0.2	*I. setosa*
4.8	3.4	1.6	0.2	*I. setosa*



Data science from engineering perspective

- It’s still software!
  - don’t forget best practices: version controlling, thorough documentation, folders structure, deployment, CI, code reviews
- Great data scientist is not necessarily great software developer
  - might not know best coding practices, how to build production ready code, how to maintain it
  - will need help from your IT department

Plan your folder structure

Template from Cookiecutter Data Science

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements
│                         generated with `pip freeze > requirements.txt
```

https://drivendata.github.io/cookiecutter-data-science/

Executing

Execution - steps

- Getting data
- Preprocessing data
- Building first (baseline) model
- Automating: building solid pipeline to make each experiment reproducible
- Keep improving model until reaching goal
- Putting in production

Collecting data and labeling

Collecting data

- How much data? The more the better :)
  (with exceptions, e.g. transfer learning)
- "Effortless collection"
- How easy it is to collect data I need for my problem

Labeling data

- Done by human ; might be easier/faster than you think
- Sometimes data is already "labeled" e.g. product recommendation

Data cleanup and preprocessing

- You might spend here much more time than you think (90/10 rule)
- Automate!
  - make it easy for you and other project participants

Building baseline model

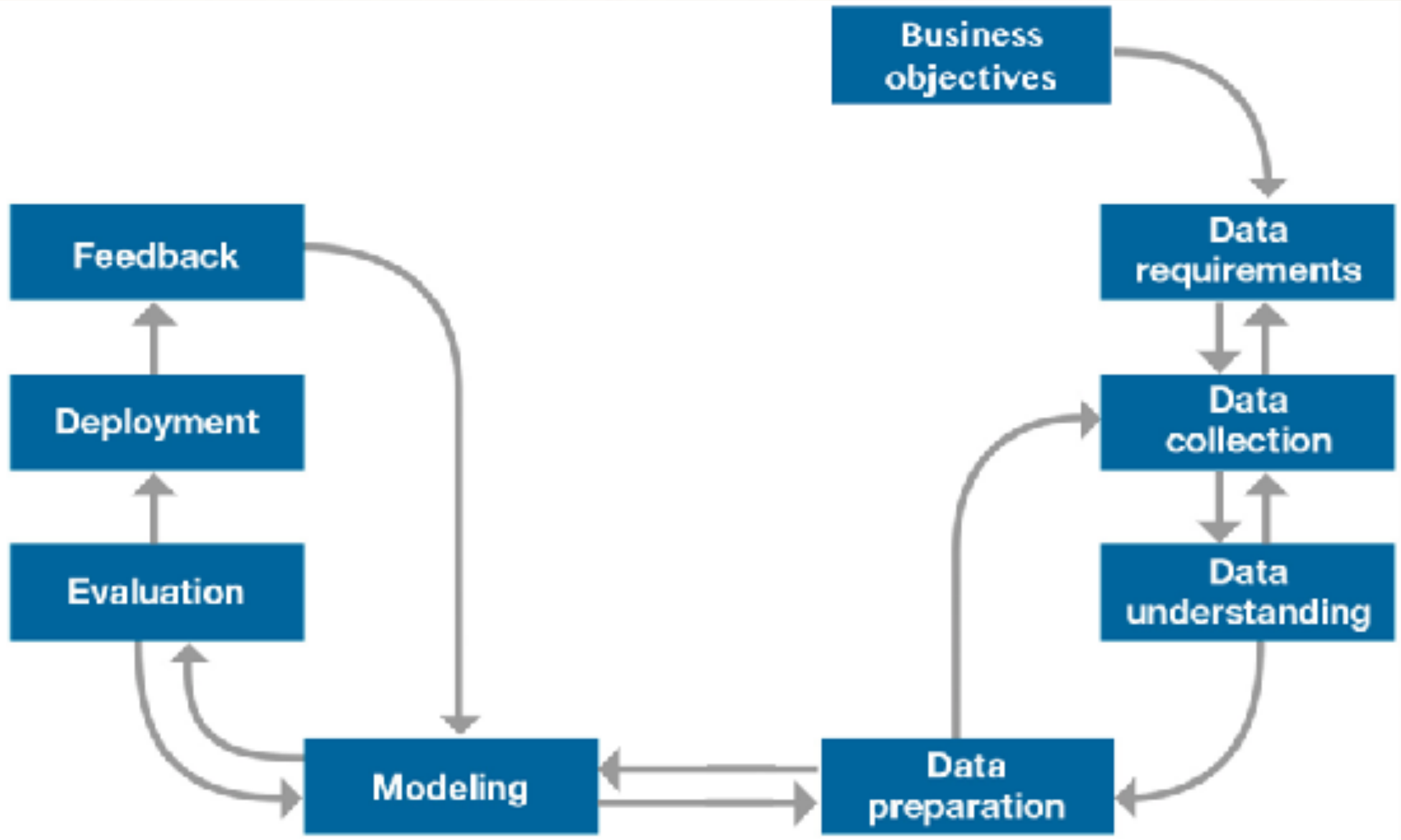
- Don't waste too much time here
- Use as a reference
- What current state-of-art can you quickly copy?

Reproducibility of your experiments

- Autoencoder in TF - v1.ipynb
- Autoencoder in TF - v2.ipynb
- Autoencoder in TF - v3.ipynb
- CLASS WEIGHTS.ipynb
- COS DIST v1 (TensorFlow).ipynb
- final Baseline TensorFlow model v1.ipynb
- final Baseline TensorFlow model v2.0 COS DIST.ipynb
- final Baseline TensorFlow model v3 COSINE DISTANCE.ipynb
- Final model v1(TensorFlow, Autoencoder based).ipynb
- Keras v1.ipynb
- Keras v2.ipynb
- Keras v3.ipynb
- Keras v4.ipynb

Living pieces

Each step can change one of the pieces in your pipeline:



- new data available
- improved way to clean data
- new insights from data
- new hypothesis modifies model structure
- … or just testing different model parameters

Reproducibility

- **Ideal scenario**: with the "push of a single button" anyone in the project can get similar experiment results ; Any experiment you've ever executed
- For each past experiment, make it easy for you and everyone else to use <u>similar</u>:
  - **data** - training/test/valid data, preprocessed results, training results (saved models)
  - **code** (preprocessing, model, validation etc.)
  - **environment** - underlying libraries, e.g. different experiment results on python2 and python3
  - **decisions** - each hypothesis should be documented ; why you're doing it, experiment outcome etc.

Build automated pipeline

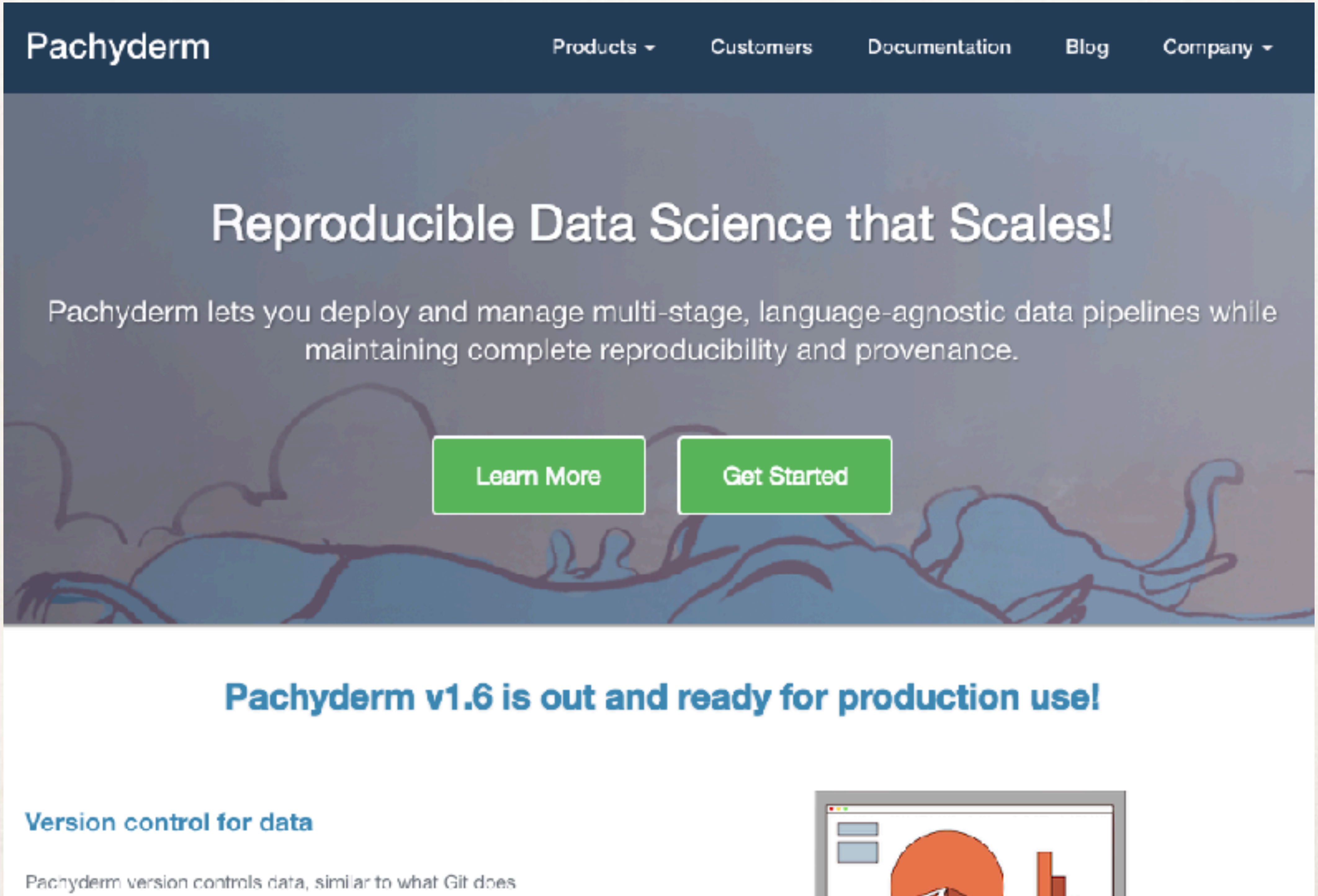
- Solutions:
  - Simpler - Python/Bash scripts
    - Each script is responsible for separate part of pipeline
    - makes it easy to substitute any part
    - Use e.g. jupyter notebook or git versioning to document each experiment, tool that puts all parts of the pipeline together
  - Advanced - Docker
    - Kubernetes - container orchestration
    - Don't try it at home :)
- Will take time and additional effort, but later you will save a lot of time while running new experiments

Doker components

```
{
  "Commit":"d6cd1e2bd19e03a81132a23b2025920577f84e37",
  "Environment":"gcr.io/docker_image_with_our_project",
  "Input_Data": "gs://google_storage_bucket/in/",
  "Output_Data": "gs://google_storage_bucket/out/"
  "Model": "CNN.py"
  "Parameters": {
      "Learning_rate": 0.03,
      "Batch_size": 100,
      .....
      "Hidden_layers": 2
  }
}
```

- For each experiment it versions your:
  - environment
  - each change in collected and preprocessed data
  - your model and model parameters
  - experiment outcome

Pachyderm http://pachyderm.io



Reproducibility - Decisions

- Use “Lab notebook” that documents how you came to the decisions that shaped your analysis
  - Explain why you’re doing it
- For simpler projects:
  - jupyter notebook
  - git versioning/branches
  - GitHub issues (with appropriate tags)

Putting in production

- Building application to serve the data
  - requires separate presentation :)

After release

Your model in production

- Maintenance
  - best practices still apply - monitoring, support, SLA and fixing bugs, etc.
  - monitoring dashboard - what average score (e.g. confidence level) am I getting on real-world data)
  - collect the data to be able to reproduce results when something goes wrong
- Model versioning
  - You might need to serve multiple versions at the same time

Thank you

Questions?