



IMT Atlantique

Bretagne-Pays de la Loire

École Mines-Télécom

Towards Production-Ready ML

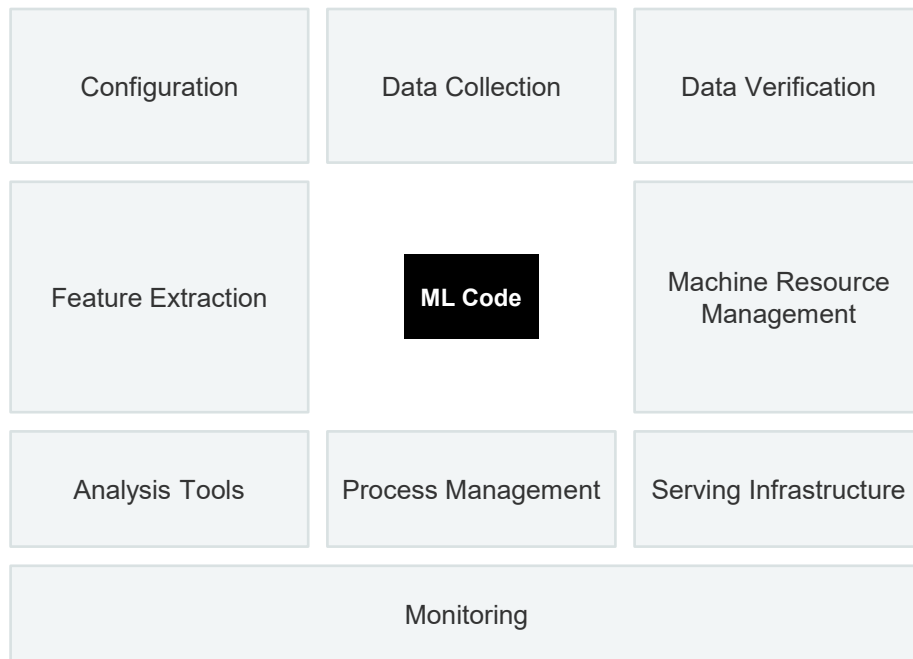
Data Science Toolkit & Applications
Lina Fahed & Laurent Brisson

SESSION 1



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

From Notebook to Production



Redrawn from Sculley et al., Hidden Technical Debt in ML Systems (NeurIPS 2015)

A real ML system is much more than a model.

The code that trains the model is only a small part of what makes it work, day after day, for other people.

Key question: can someone else rebuild, understand and rerun your result?

Writing the model is the easy part.

Four Pillars of Production-Ready ML

1

Reproducible Environment

Anyone can rebuild exactly the same setup, on any machine.

2

Versioned History

Every change is recorded, explained and reversible.

3

Structured & Tested Code

Logic lives in reusable functions, checked by automated tests.

4

Reliable Pipeline

Evaluation reflects performance on data the model has truly never seen.

Reproducible Environment

VS Code + WSL

One workspace, the same Linux environment for everyone.

uv

Installs Python, declares and locks every dependency.

Git / GitLab

Keeps the history and shares the project.

File	Role	In Git?
<code>pyproject.toml</code>	Declares the project and its direct dependencies	✓ Yes
<code>.python-version</code>	Pins the expected Python version	✓ Yes
<code>uv.lock</code>	Records the exact resolved versions of every package	✓ Yes
<code>.venv/</code>	Local environment, rebuilt with <code>uv sync --locked</code>	✗ No

Goal: end “it works on my machine”. Git stores the recipe, uv rebuilds the kitchen.

Git as the Memory of the Project

Good practices

- **One commit** per logical step, not one big pile at the end
- **Readable messages** that say why the change exists
- **A shared history** that shows how the project was built

Conventional Commits

```
type(scope): short description
```

```
feat · fix · refactor · test · docs · chore
```

```
$ git log --oneline  
a3f9c21 docs: document main findings  
7be1d04 feat(eval): cross-validate  
52c8e90 test(data): cover threshold  
1d0a6f3 refactor: extract data.py  
9e4b2a7 chore: initialize project
```

Read from bottom to top, the log tells how the project was built, even months later.

SESSION 2



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Where Does It Fit?

CRISP-DM Steps:

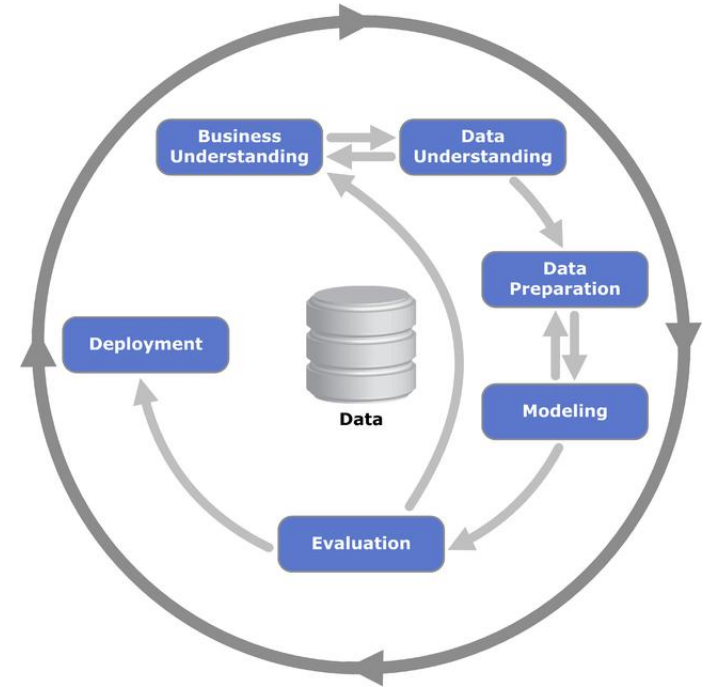
1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. **Deployment** (where production starts)

Production-readiness is **not a final step**: it is prepared from the very first line of code.

Session 1 environment and Git

Session 2 first pipeline, turned into a tested package

Session 3 a robust, tracked, reusable pipeline



Four Pillars of Production-Ready ML

1

Reproducible Environment

Anyone can rebuild exactly the same setup, on any machine.

2

Versioned History

Every change is recorded, explained and reversible.

3

Structured & Tested Code

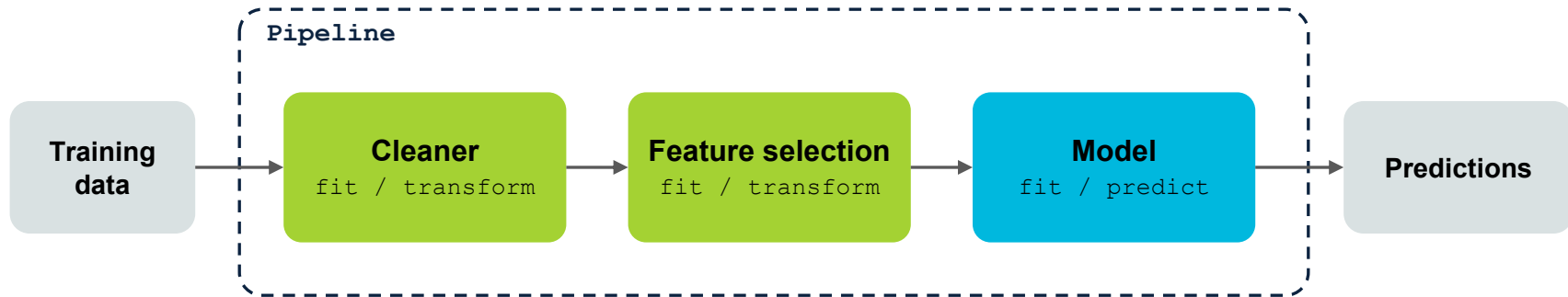
Logic lives in reusable functions, checked by automated tests.

4

Reliable Pipeline

Evaluation reflects performance on data the model has truly never seen.

The Scikit-learn Pipeline



One contract ✓

Every step exposes the same methods: `fit`, then `transform` or `predict`.

Your own steps and scikit-learn's built-in ones plug together like building blocks.

No data leakage ⚠

Some steps **learn** something: which columns to drop, which features to keep.

Inside a pipeline, they learn **only from training data**, never from the data used to evaluate.

Notebook vs Package

	Notebook 	Package 
Purpose	Explore, try ideas, look at data	Run the pipeline reliably
State	Hidden, depends on cell order	Explicit inputs and outputs
Reuse	Copy-paste cells	Import functions
Verification	Rerun and look	Automated tests (pytest)

The notebook explores, the package runs.

```
air_quality/  
├── pyproject.toml  
├── notebooks/ explore  
├── src/air_quality/  
│   ├── data.py  
│   ├── features.py  
│   ├── evaluation.py  
│   └── workflows.py map  
├── tests/ verify  
└── scripts/ run
```

Git as the Memory of the Project

Good practices

- **One commit** per logical step, not one big pile at the end
- **Readable messages** that say why the change exists
- **A shared history** that shows how the project was built

Conventional Commits

```
type(scope): short description
```

```
feat · fix · refactor · test · docs · chore
```

```
$ git log --oneline  
a3f9c21 docs: document main findings  
7be1d04 feat(eval): cross-validate  
52c8e90 test(data): cover threshold  
1d0a6f3 refactor: extract data.py  
9e4b2a7 chore: initialize project
```

Read from bottom to top, the log tells how the project was built, even months later.

SESSION 3



IMT Atlantique
Bretagne - Pays de la Loire
École Mines-Télécom

Where Does It Fit?

CRISP-DM Steps:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. **Deployment** (where production starts)

Production-readiness is **not a final step**: it is prepared from the very first line of code.

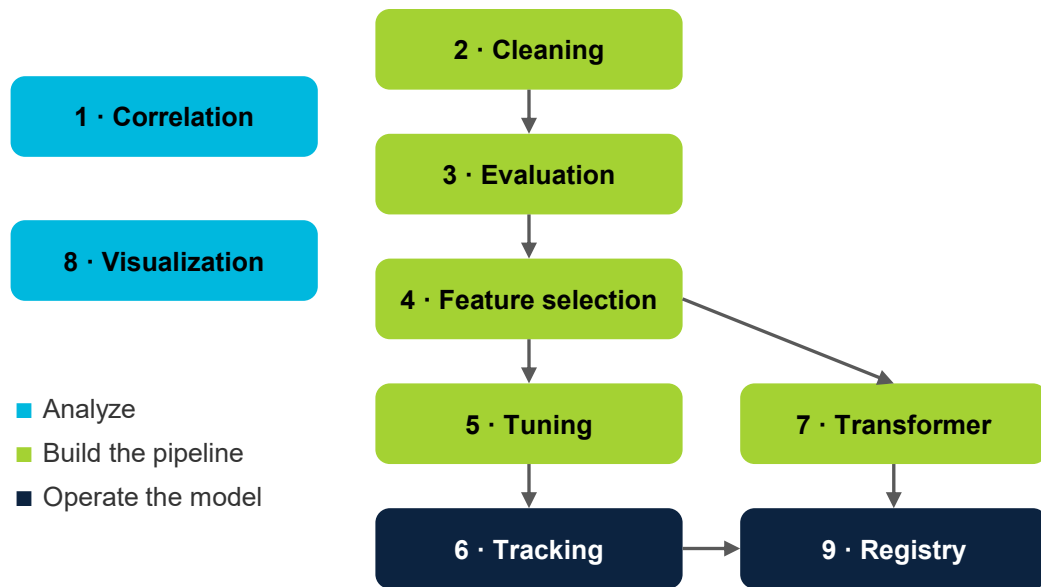
Session 1 environment and Git

Session 2 first pipeline, turned into a tested package

Session 3 a robust, tracked, reusable pipeline

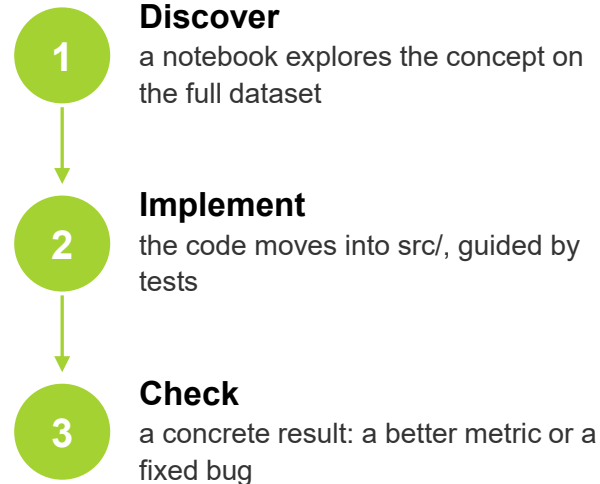


The Big Picture: Session 3



Free order: just respect the prerequisites)

Every module, the same loop



Git as the Memory of the Project

Good practices

- **One commit** per logical step, not one big pile at the end
- **Readable messages** that say why the change exists
- **A shared history** that shows how the project was built

Conventional Commits

```
type(scope) : short description
```

```
feat · fix · refactor · test · docs · chore
```

```
$ git log --oneline  
a3f9c21 docs: document main findings  
7be1d04 feat(eval): cross-validate  
52c8e90 test(data): cover threshold  
1d0a6f3 refactor: extract data.py  
9e4b2a7 chore: initialize project
```

Read from bottom to top, the log tells how the project was built, even months later.

What to Remember



Reproducible: uv and a lockfile: anyone rebuilds the same environment.



Traceable: Git history says what changed, when, and why.



Tested: Reusable functions verified by pytest, not hidden notebook state.



Reliable by construction: A scikit-learn Pipeline learns only from training data.

Writing the model is the easy part.

Making it **reproducible, understandable and reusable** is what turns an experiment into a result others can trust.