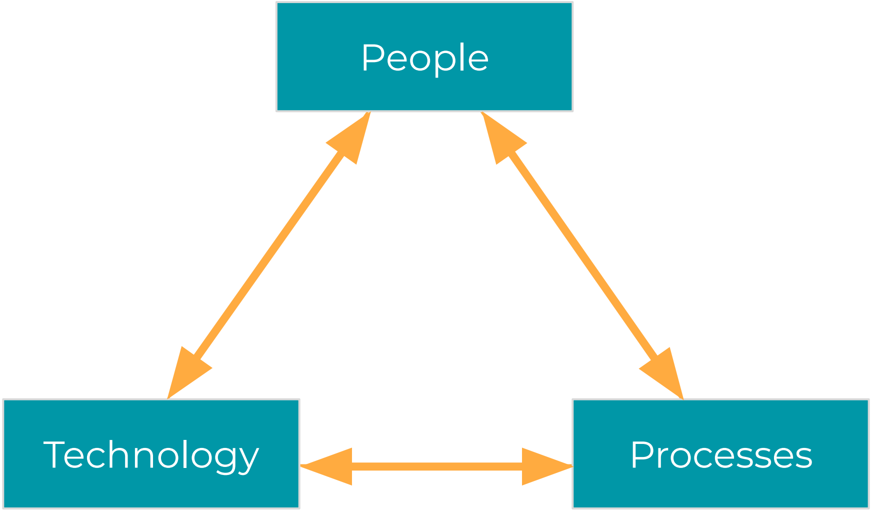
Wit Jakuczun @ WLOG Solutions

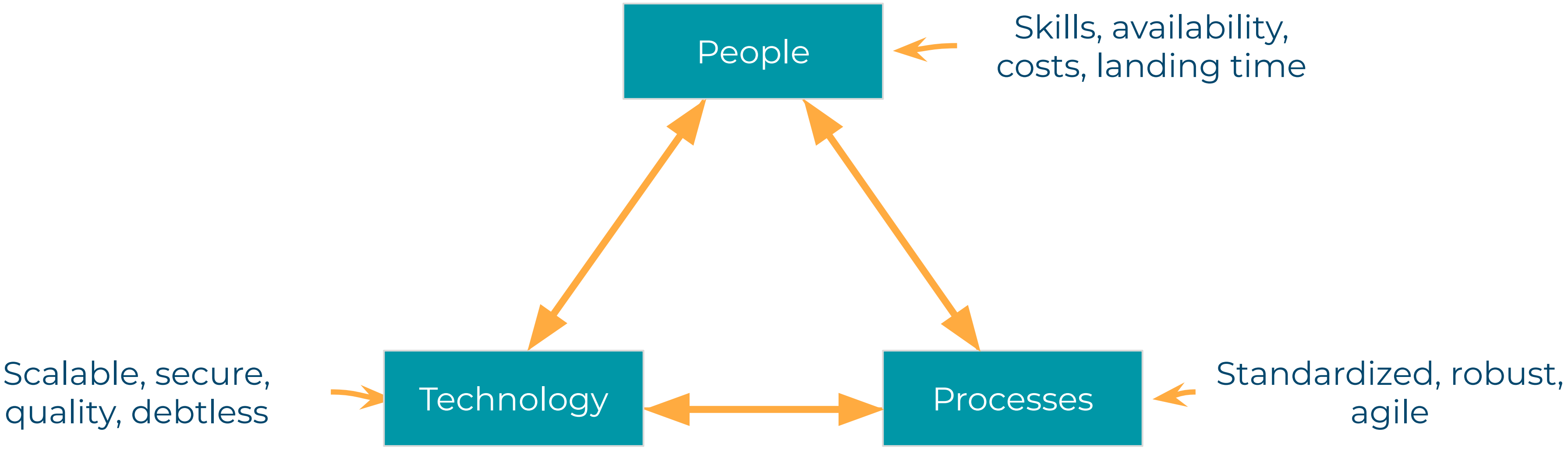
Effective Data Science process

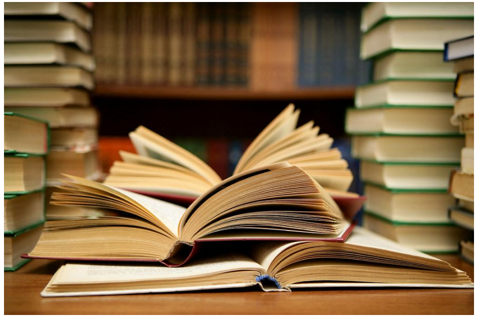
ABD point of view

Effective data science is a conjunction of three aspects

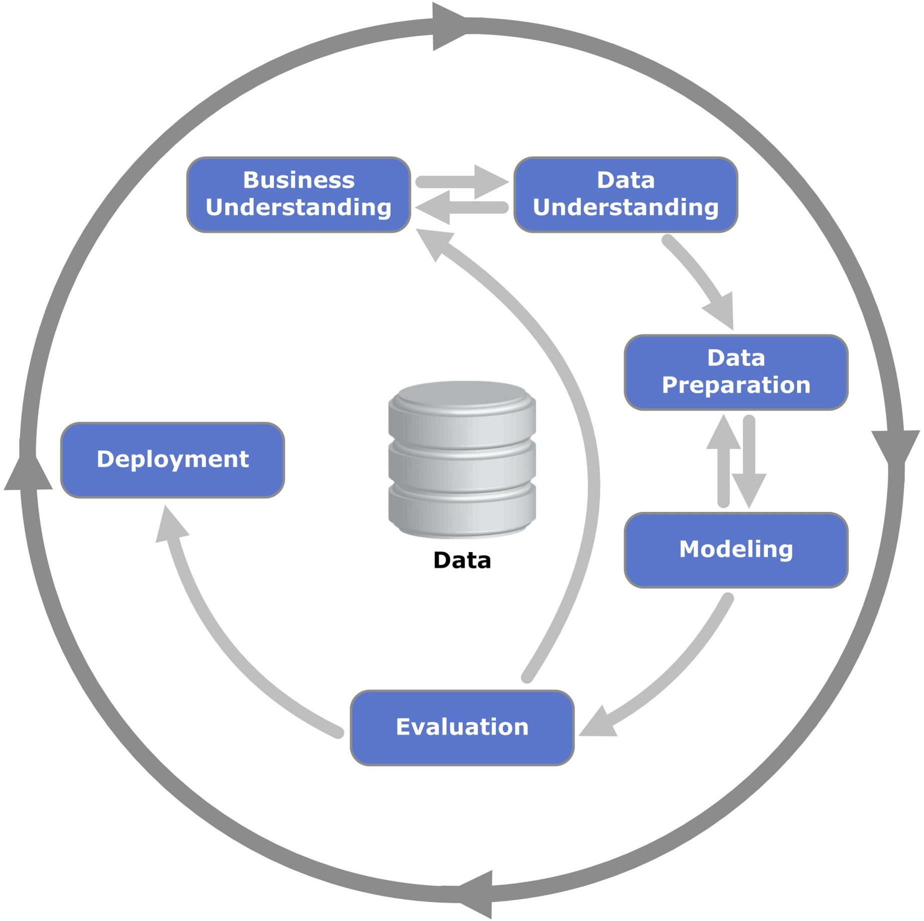


Effective data science is a conjunction of three aspects



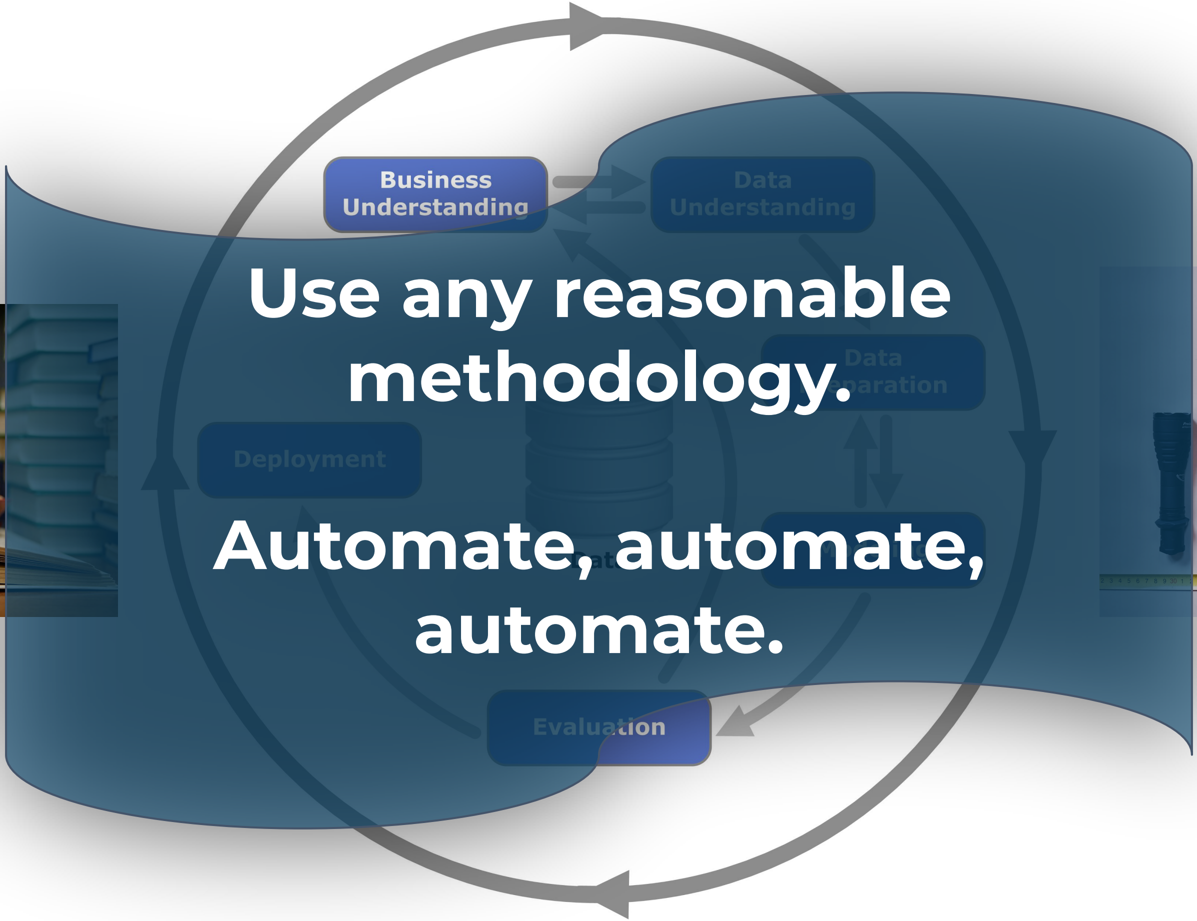


Knowledge



Tools

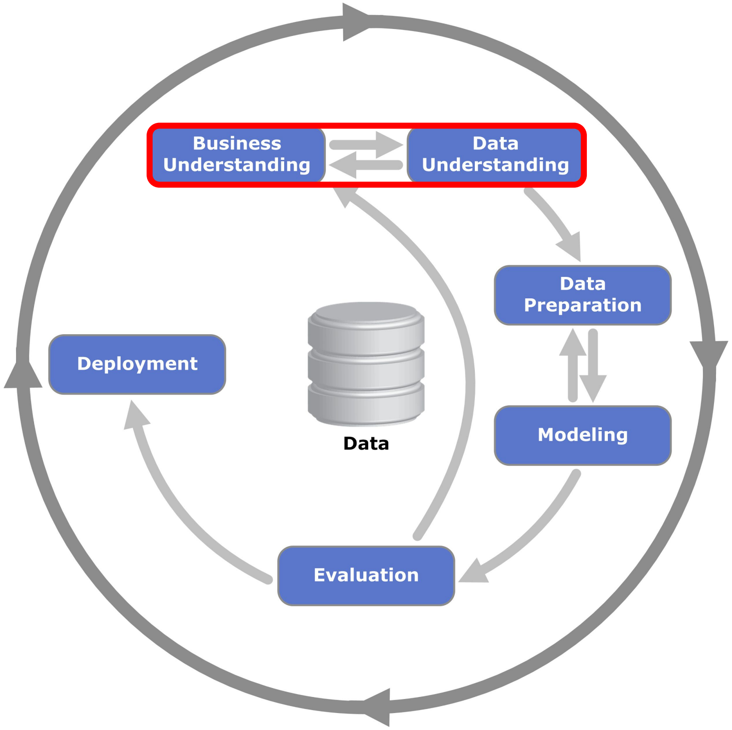
Knowledge



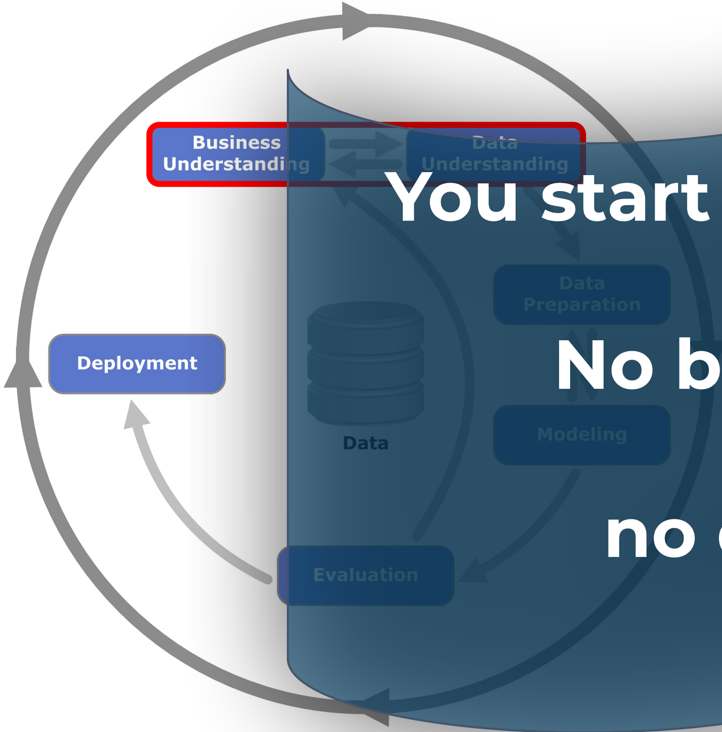
Use any reasonable methodology.

Automate, automate, automate.

Tools



1. What is a business case for a model?
2. When exactly the model is being run?
3. What data is available at the moment?
4. When and how is the recommendation using score generated?
5. When do you get the feedback - how good was the recommendation?
6. What is a test plan for the evaluation phase?
7. Is all the data historized?



1. What is a business case for a model?
2. When exactly the model is being run?
3. What data is available at the moment?
4. When and how is the recommendation using score generated?
5. When do you get the feedback - how good was the recommendation?
6. What is a test plan for the evaluation phase?
7. Is all the data historized?

You start deployment here!

No business value
==
no deployment

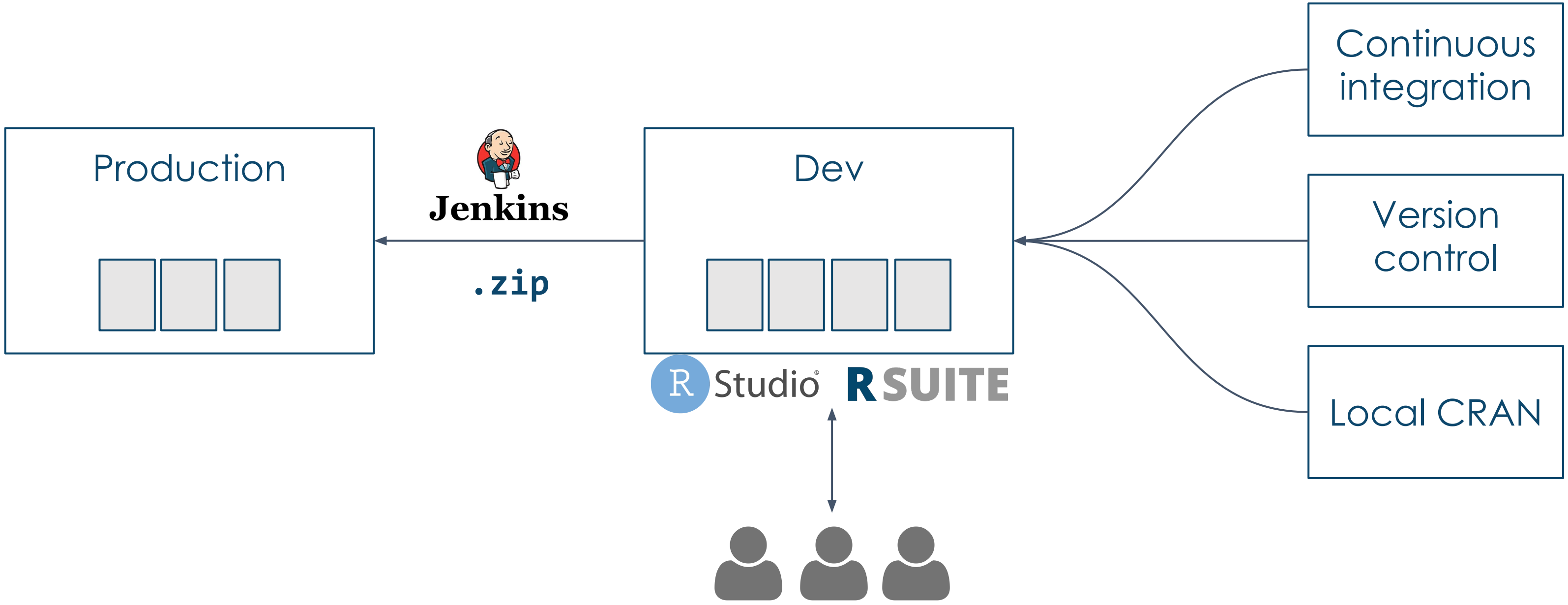
R development & deployment

Software engineer point of view

POC vs Deployment

	POC	**Productional**
Goal	Evaluate: Is feasible? Estimate: Is it worth? Response can be **No**	Deliver functionality
User	Internal / Developer	External / Non developer
Environment	Development	Production
Lifetime	Short	Long

Production ready setup



Need to support

At the same time

- Development
- On production
  - Bug fixing
  - Backward support
  - Data & configuration migrations

Reproducibility is required!

Preparation for release

- Get ready for upgrade
  - How not to break consumer systems?
  - How not to break used functionalities?
  - How not to lose valuable data on production?
  - What version is deployed on production?
- Get ready for support: in case of problem
  - Is it possible to detect cause?
  - Is it possible to reproduce?
  - Is it possible to fix without upgrade?

Conclusion

To productionize analytical solution you should handle it as **any other software solution**

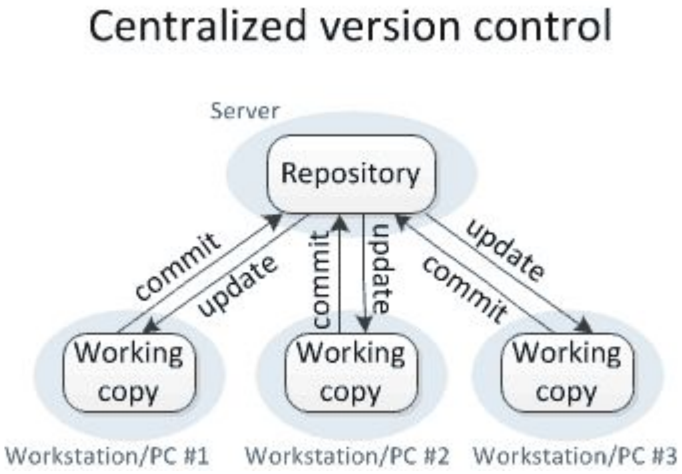
You need to use **software solution development guidelines**

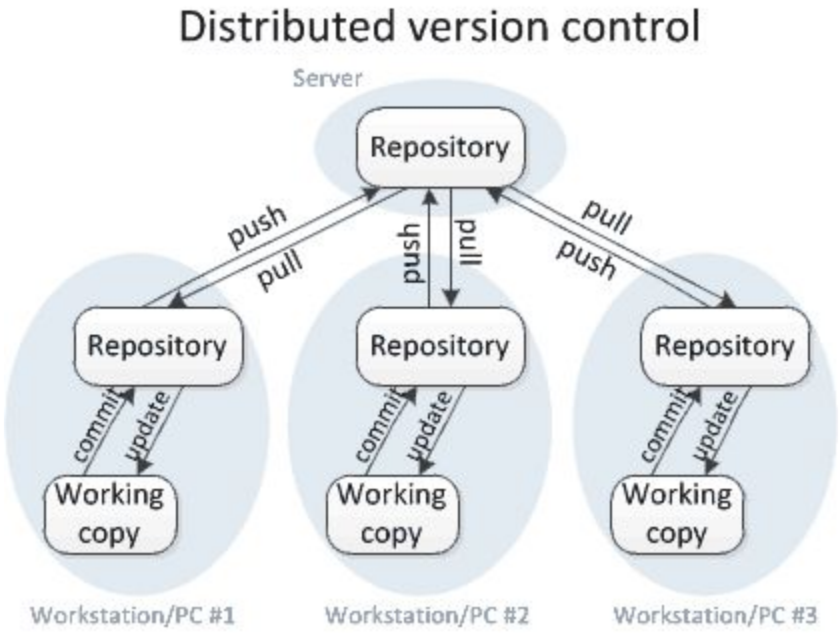
Version control - why?

- Collaboration
  - Working on same source code
  - Merging automatization

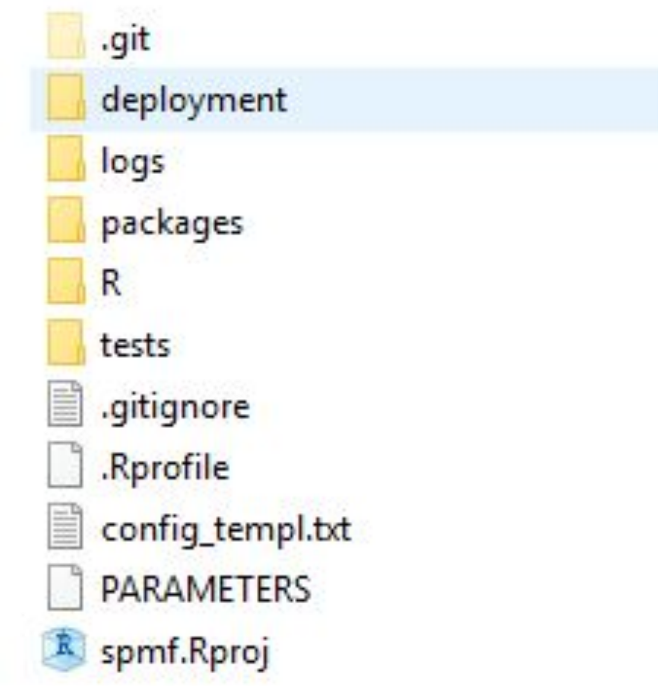
- Change history
  - Change description
  - Change log
  - “Time machine”

Version control - svn vs git





Standardized project (RSuite)



Project is integral - must be managed as a whole

- Master scripts to control a workflow
- Project local packages to control complexity of your code
- Dependencies definition to reproduce results

Why do we need standardization

- Flat learning curve
- Easier automatization
- Simpler project launch
- Support good practices from beginning

Dependency management

How to **support** Project A and **develop** Project B on same computer?

Project A

Developed in Jan 2016
Uses data.table v 1.9.6

System-wide
Packages

/usr/local/lib/R/site-library

Project B

Developed now
Uses data.table v 1.12.2

Don't forget sub-dependencies!!

Project environment encapsulation

Project A

Developed in Jan 2016
Uses data.table v 1.9.6

Project A packages

./Project_A/libs

Project B

Developed now
Uses data.table v 1.12.2

Project B packages

./Project_B/libs

That's not enough

⚠ Package versions on CRAN: come and go

Use MRAN - daily CRAN snapshots **SOLVED**

⚠ Some packages are on GitHub only

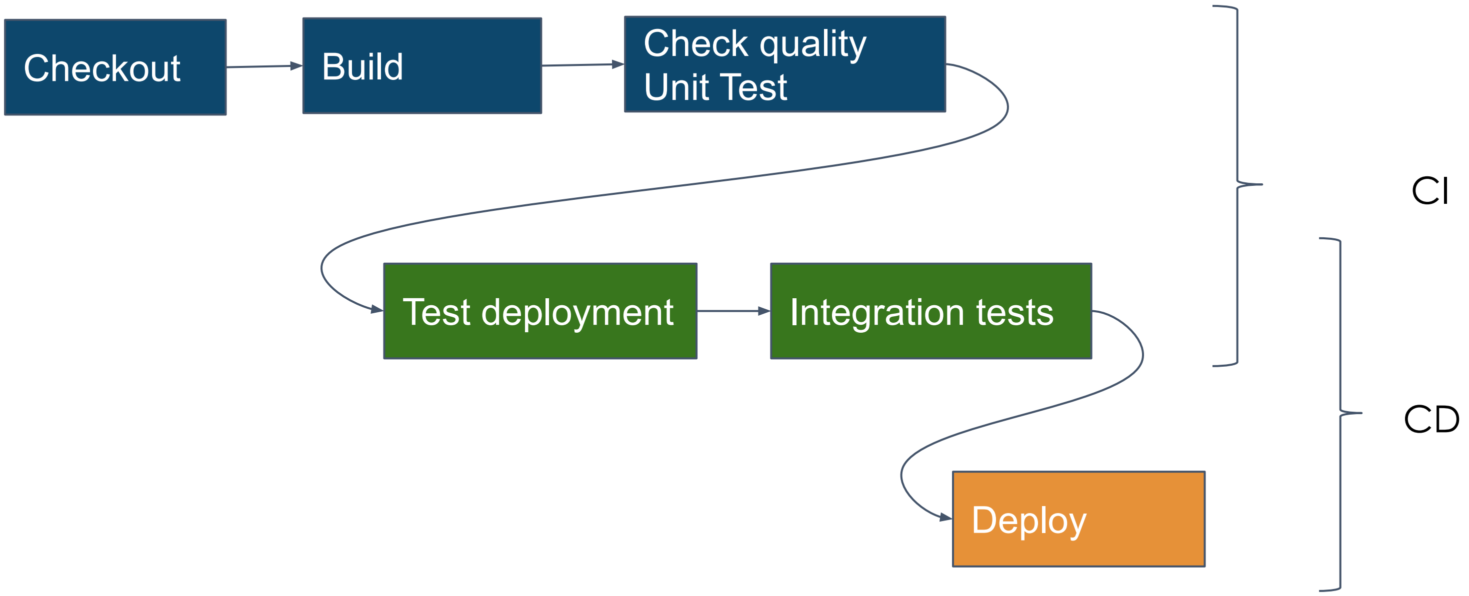
no control over versioning

Use in-house package repository **SOLVED**

Testing - what do we test

- Unit tests
  - White box - can see internals
  - Small and fast
  - Responds if code works properly
- Integration tests
  - Black box - can not see internals
  - Do systems communicate as expected
  - Can be time consuming
  - Responds if solution fits “architectural puzzle”
- Performance/Load/Stress tests

Continuous Integration/Delivery



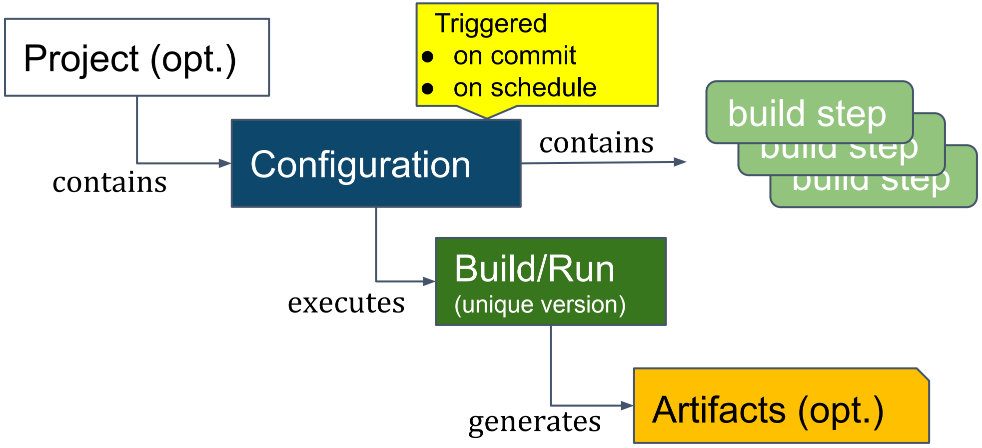
CI/CD - Why?

- Automatization
  - no human - no errors
  - clear environment - clone of prod: “works for me” issue
  - routine tasks - check every commit
  - possible: frequent integrations

- Dev/Prod mediator

CI/CD - How?

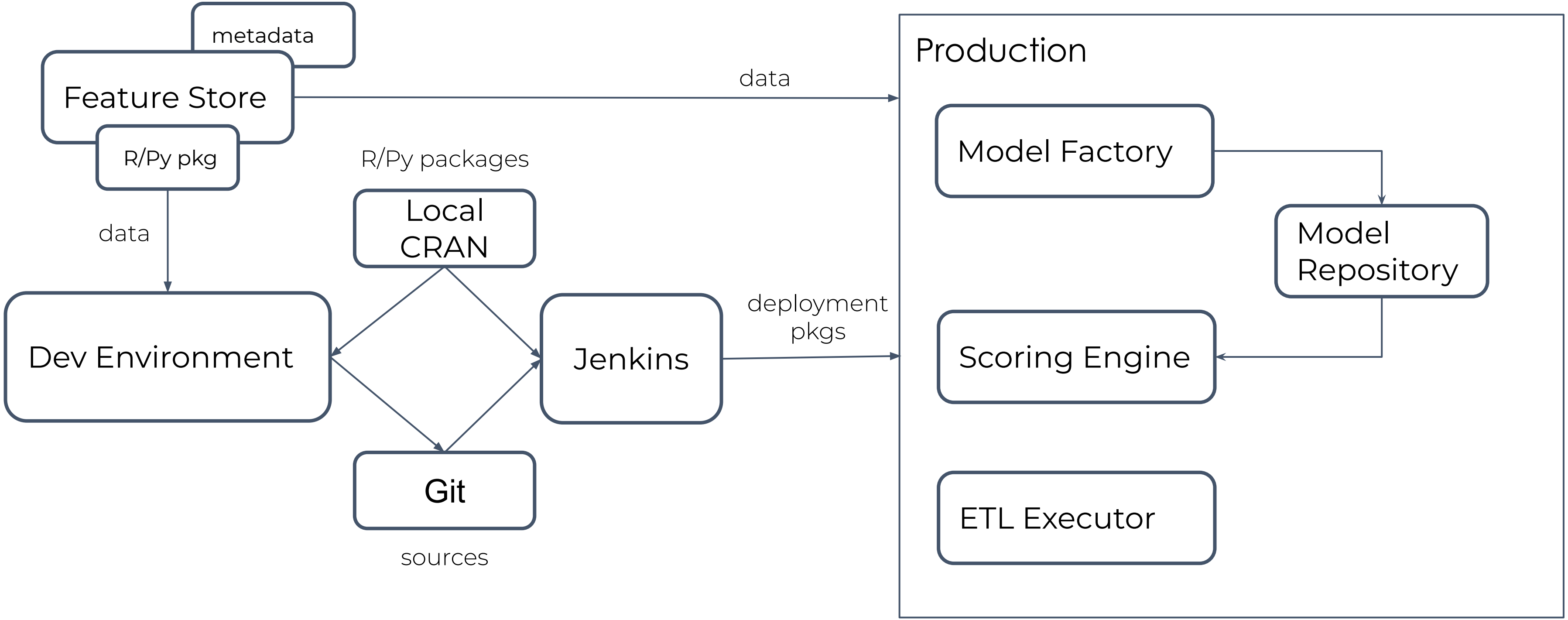
Lots of tools, much alike concept:

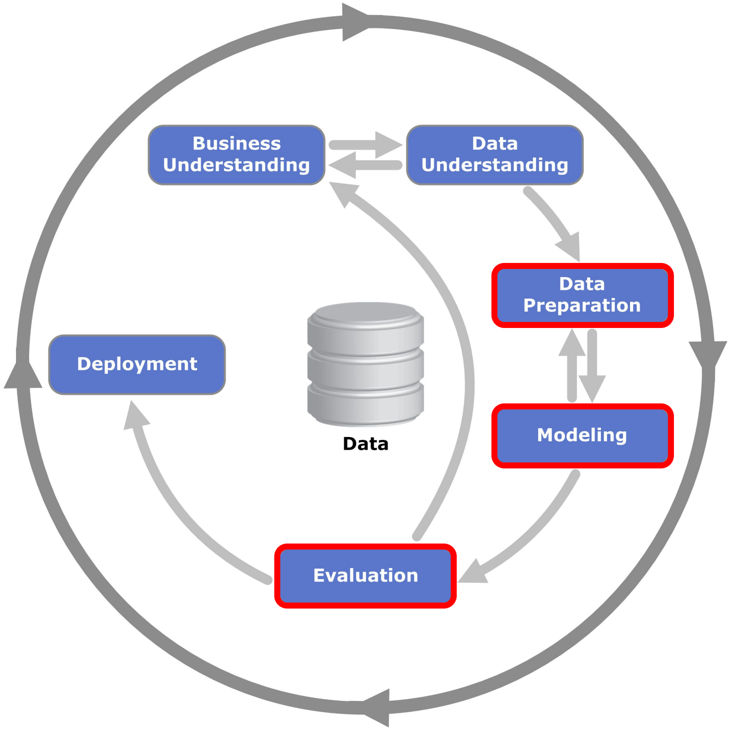


Production setup for ML

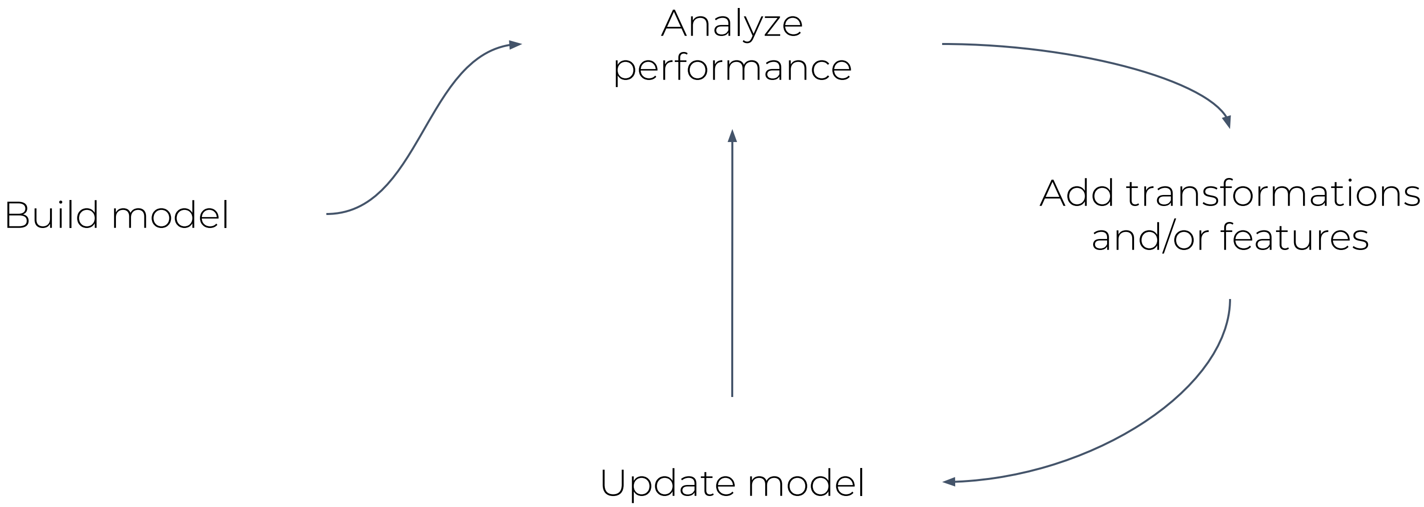
Development, Deployment, Production

Production setup for ML




Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Data

Model development process



Model development process

Analyze performance

Build your transformation library

Use abstraction to run and compare experiments.

Build model

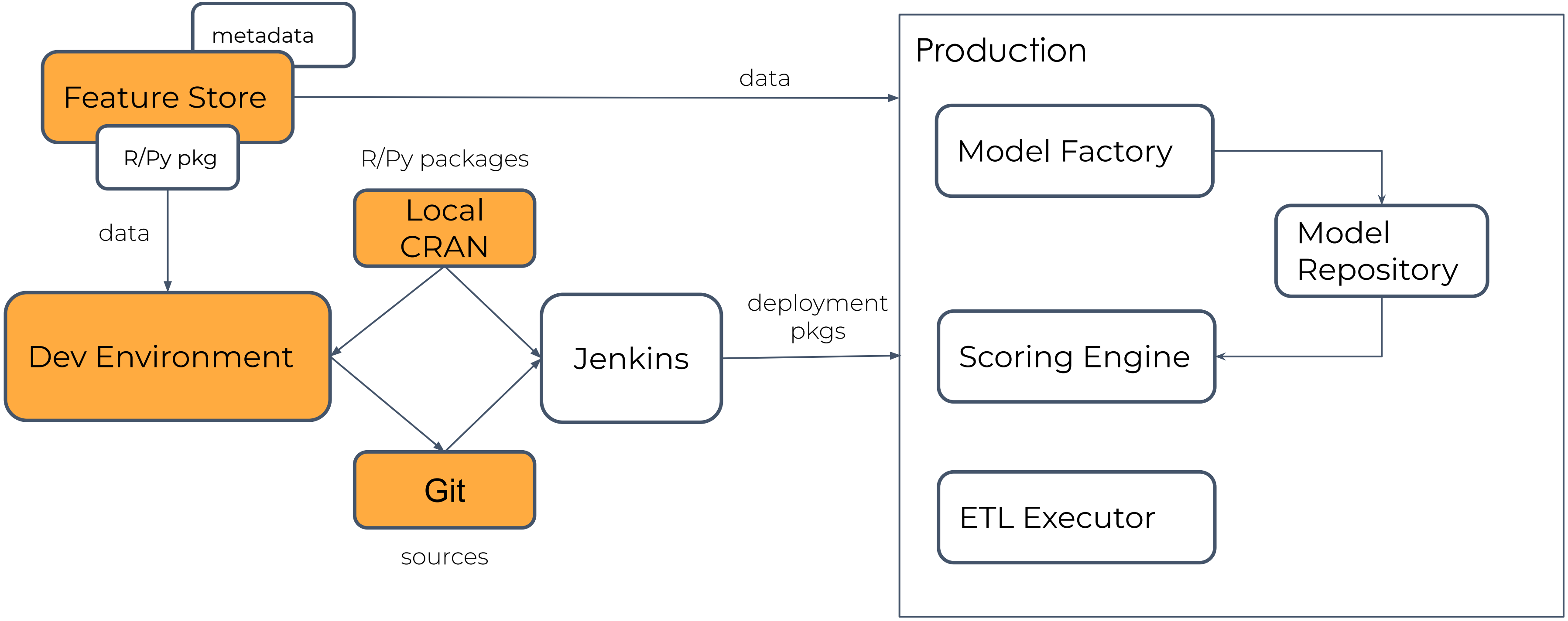
Add transformations and/or features

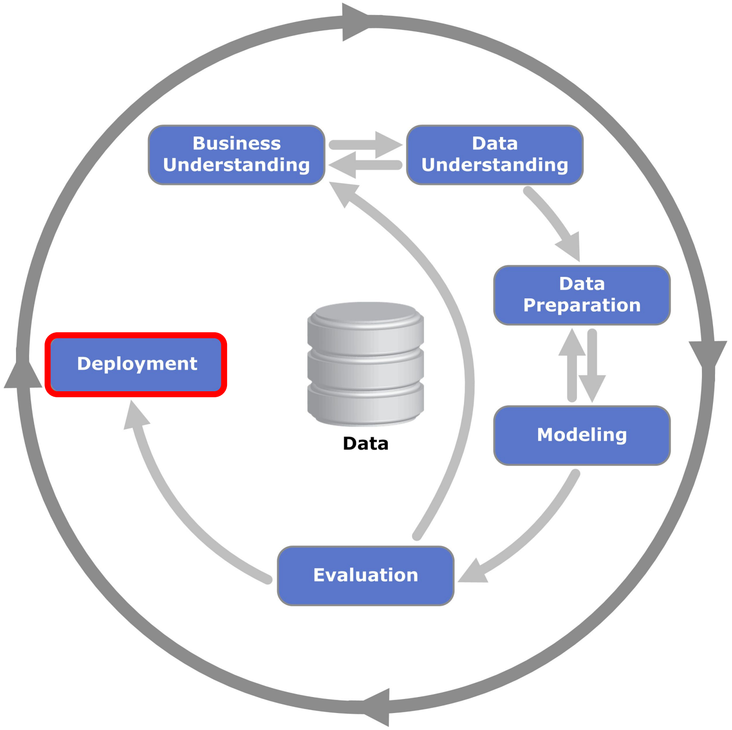
Consider AutoML for initial model.

No extra coding after EDA!

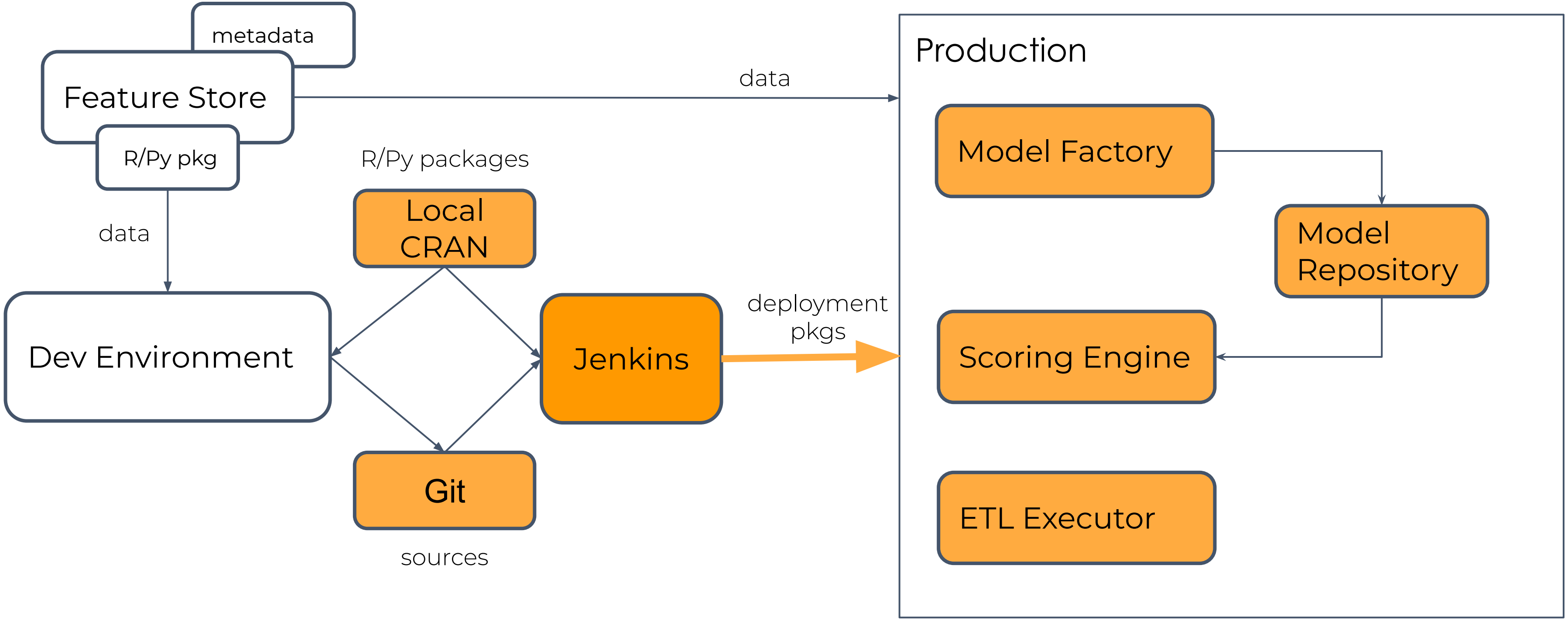
Update model

ML Development phase

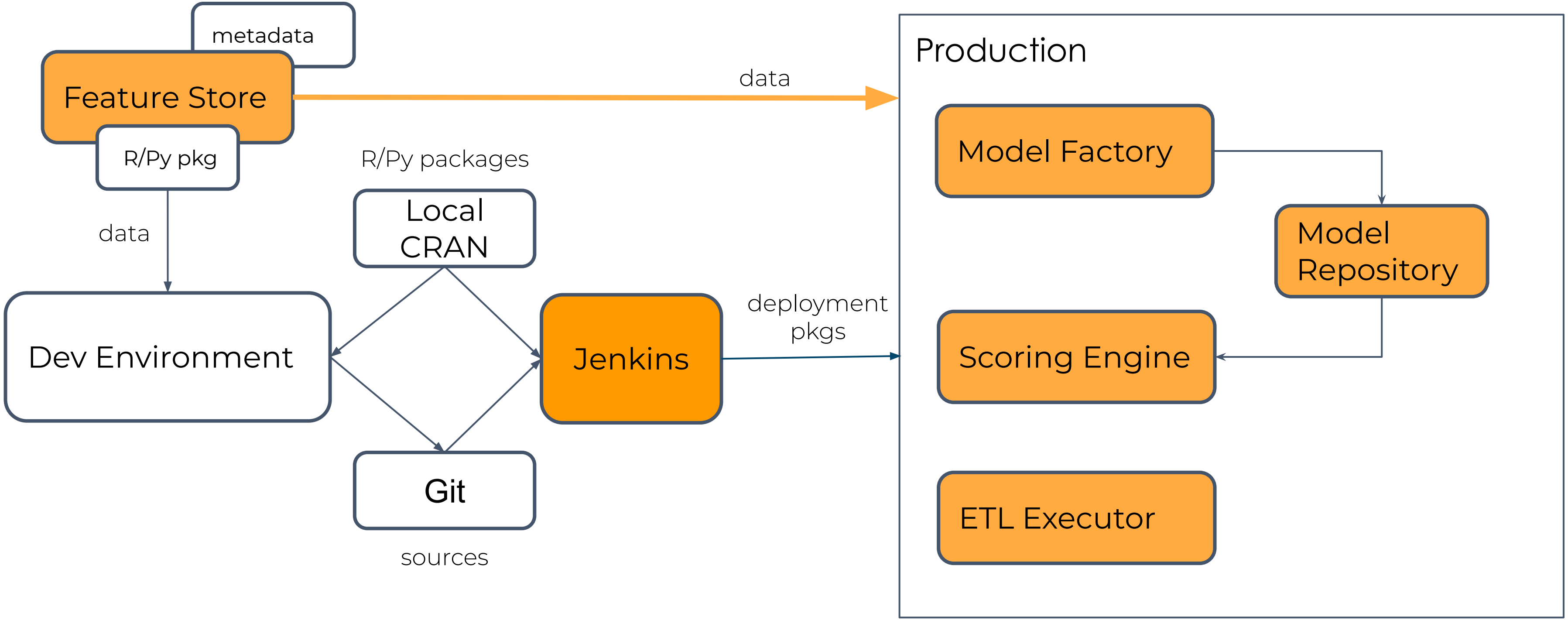


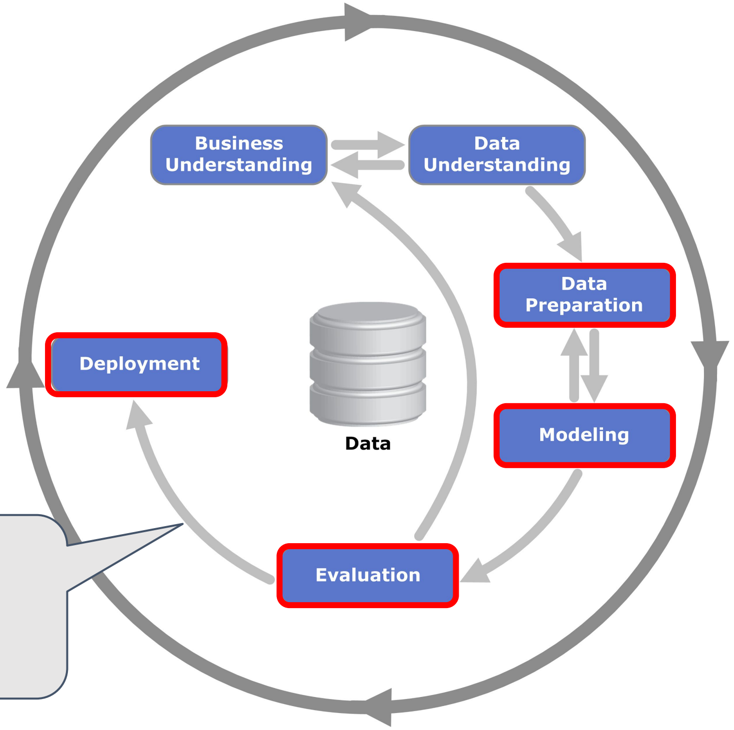

Business Understanding
Data Understanding
Data Preparation
Deployment
Data
Modeling
Evaluation

ML Deployment phase



ML "On Production" phase





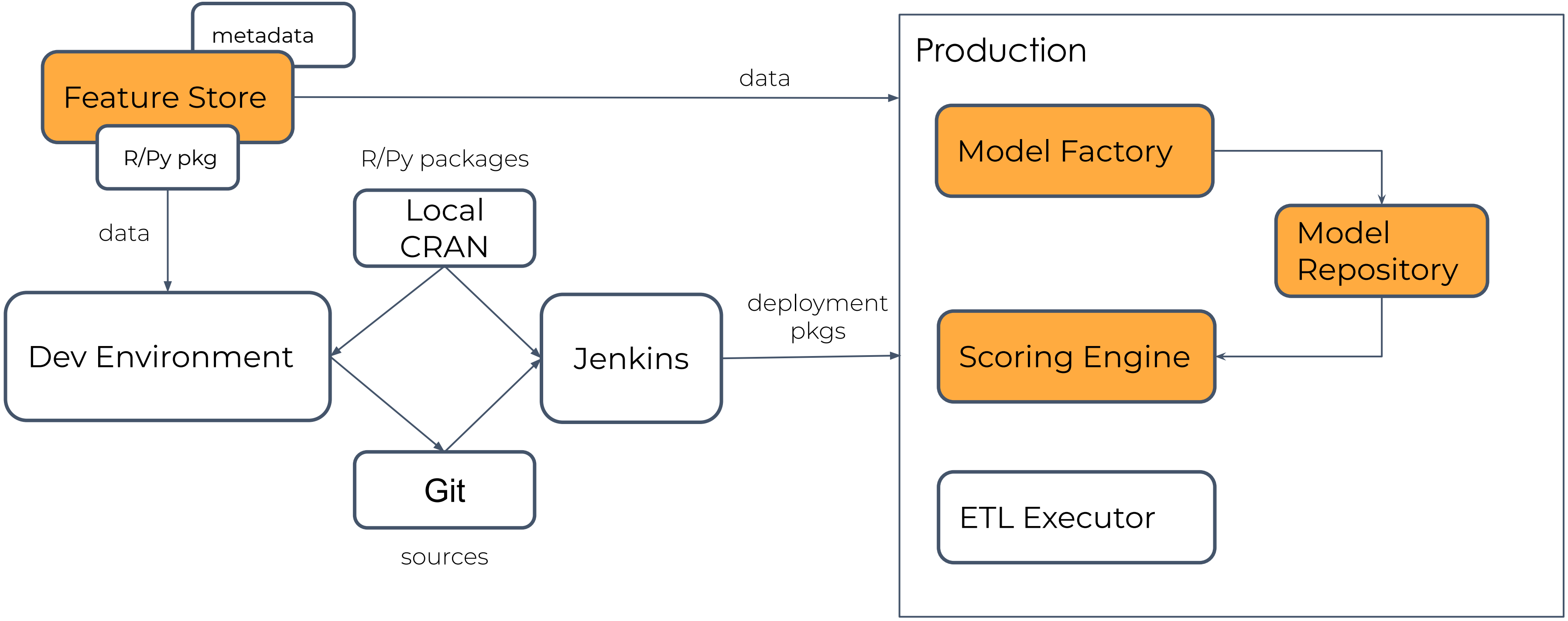
Automatic!

No coding here.
No pkg installation here.

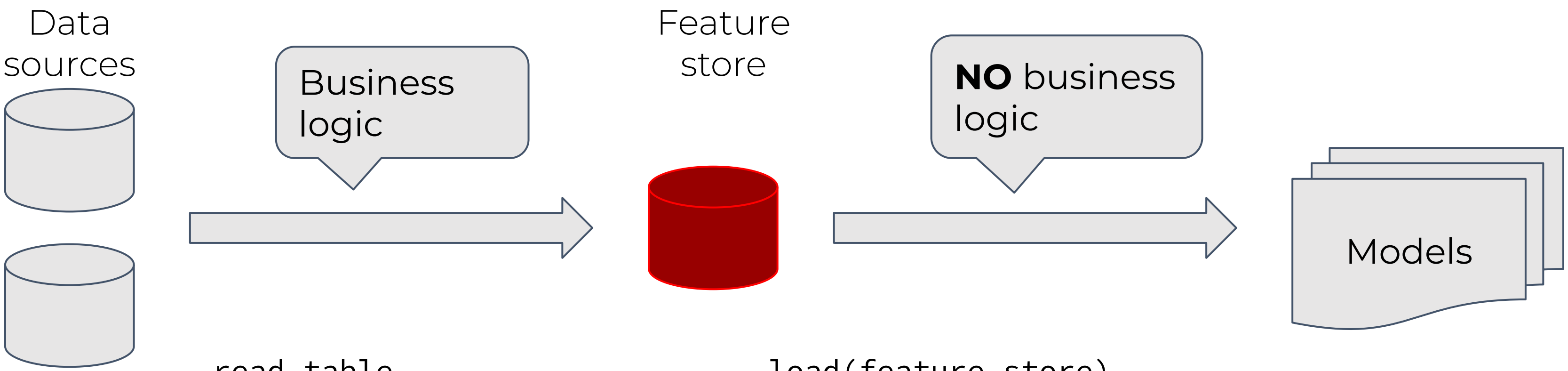
Production setup for ML

Important concepts/abstractions

Important concepts



Feature store design pattern




```
read.table
dbReadTable
```

```
load(feature_store)
dt <- get_ftrs(“customer”,
               “profile.*”,
               “response”)
```

ETL/ELT

No read.csv
No SQL

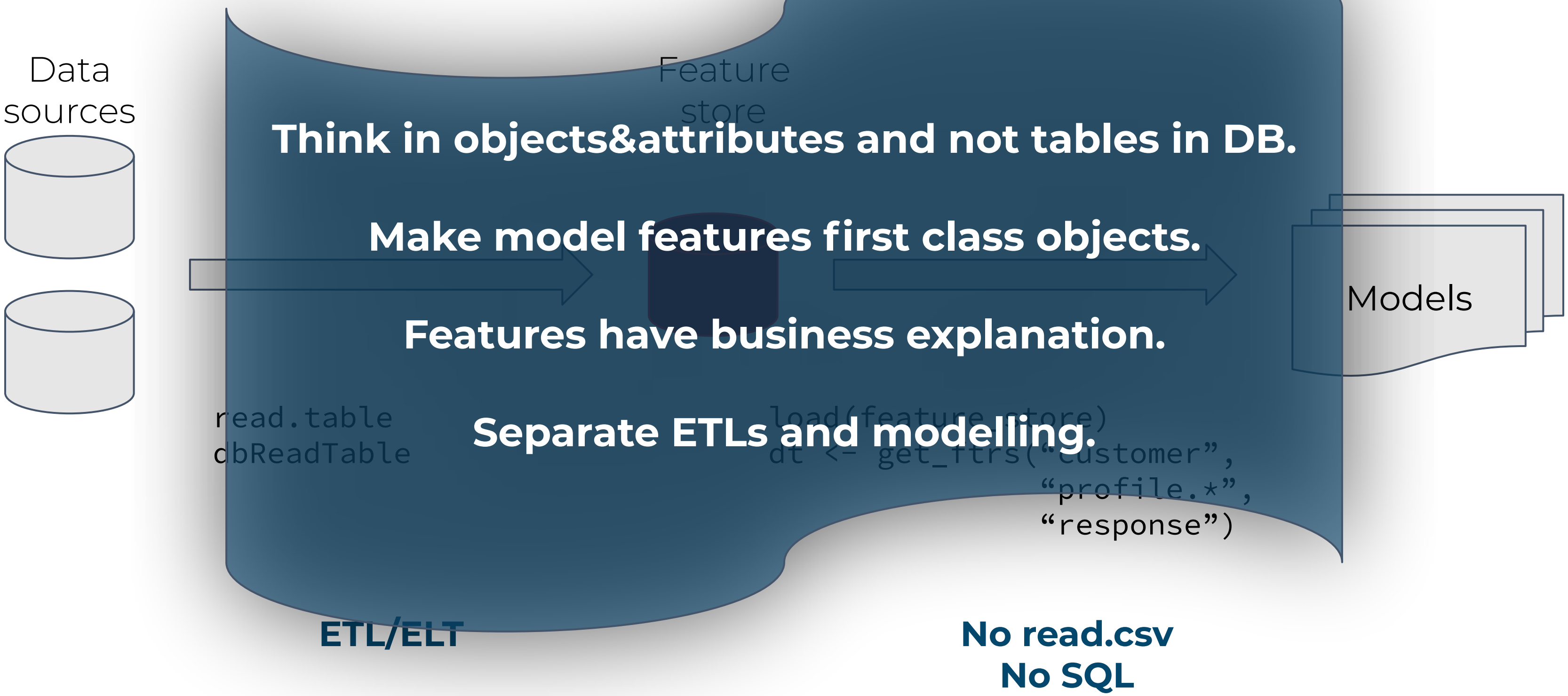
Feature store design pattern

Think in objects&attributes and not tables in DB.

Make model features first class objects.

Features have business explanation.

Separate ETLs and modelling.



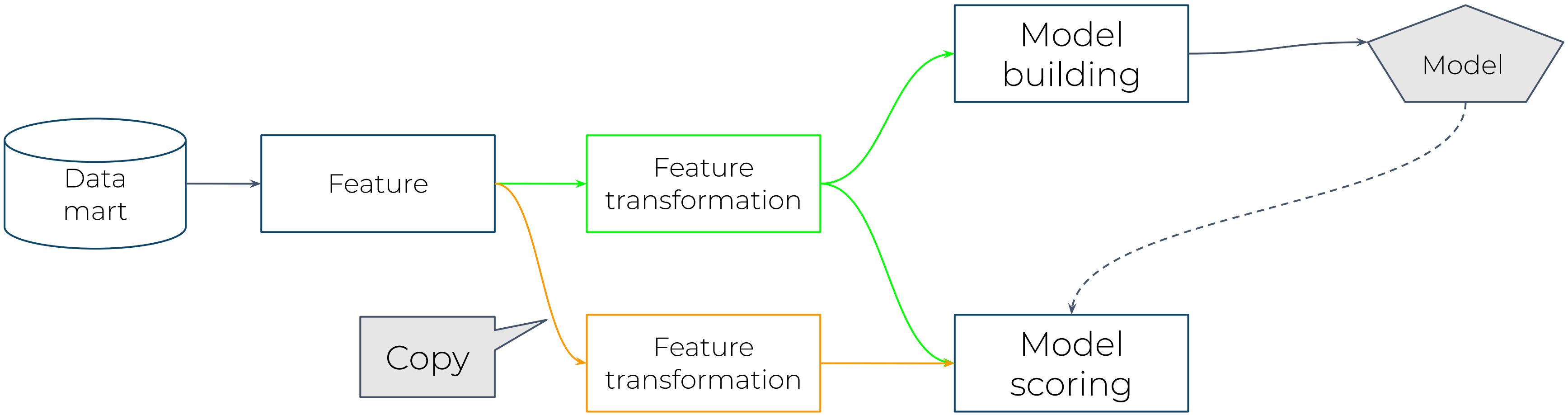
Feature store design pattern

Object Relational Mapper (ORM) design pattern.

Data Warehouse pattern.

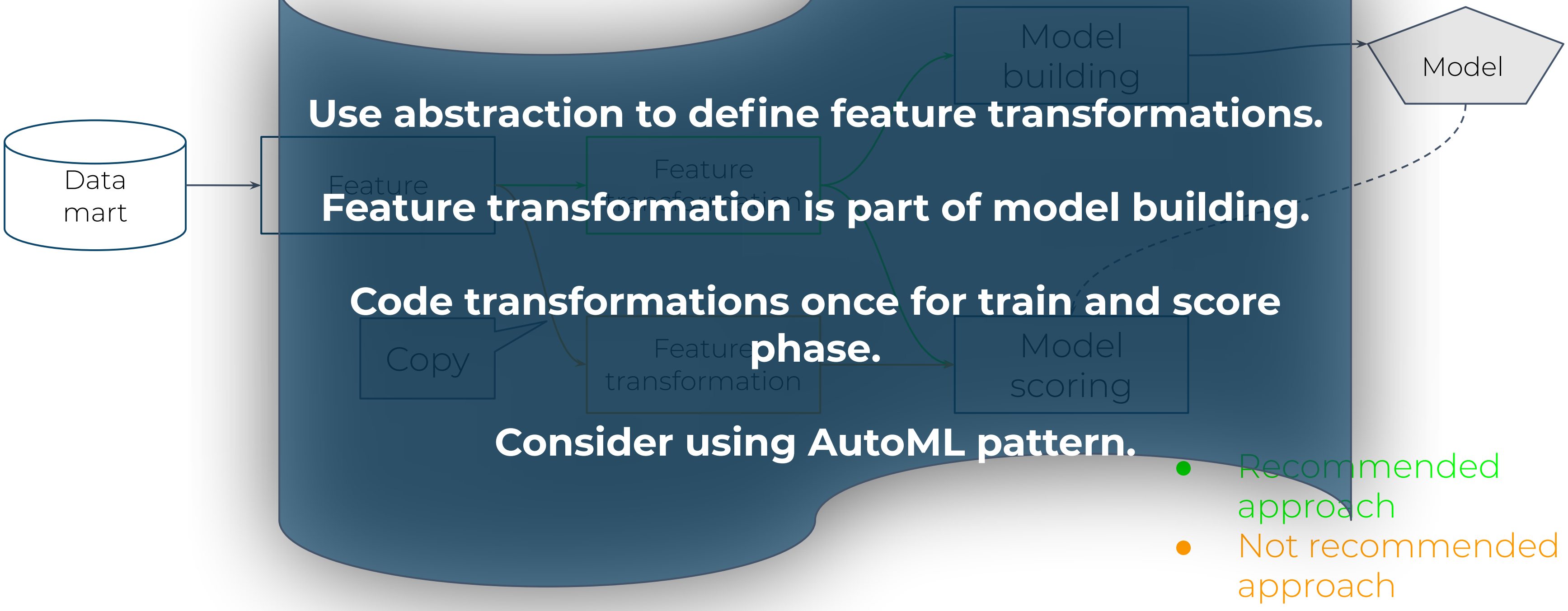
Business objects pattern.

Model factory & Scoring Engine design pattern



- Recommended approach
- Not recommended approach

Model factory & Scoring Engine design pattern



Use abstraction to define feature transformations.

Feature transformation is part of model building.

Code transformations once for train and score phase.

Consider using AutoML pattern.

Model repository

- Model A
    - Model Instance A.1
    - Model Instance A.2
- Model B
    - Model instance B.1
    - Model instance B.2
    - Model instance B.3
- ...

Model repository

- Model A
  - Model Instance A.1
  - Model Instance A.2
- Model B
  - Model inst
  - Model instance B.2
  - Model instance B.3
- ...

“Artifactory” pattern.

Models must be versionized.

No, git is not good :).

Summary

Always Be Deploying policy

- Deployment starts with business understanding phase
- Deploying R is like for any other programming language
- Deploying ML requires additional abstractions: Feature Store, Model Factory, Model Repository, Scoring Engine.

Wit Jakuczun
CEO

wit.Jakuczun@wlogsolutions.com

+48 601820620

http://www.wlogsolutions.com