

# How to Approach the Course



# Section 1

- Materials for the course
  - Code
  - Presentations
  - Datasets
- How to approach the course
- **Please don't skip this section!**

# • “Theoretical” sections 2 and 3



- Model deployment and why it matters
- Research and production environments
- Machine learning systems architecture
- Challenges and principles of designing the systems architecture
- **Practical overview – no code**



# • Section 4 - Research Environment

## First half

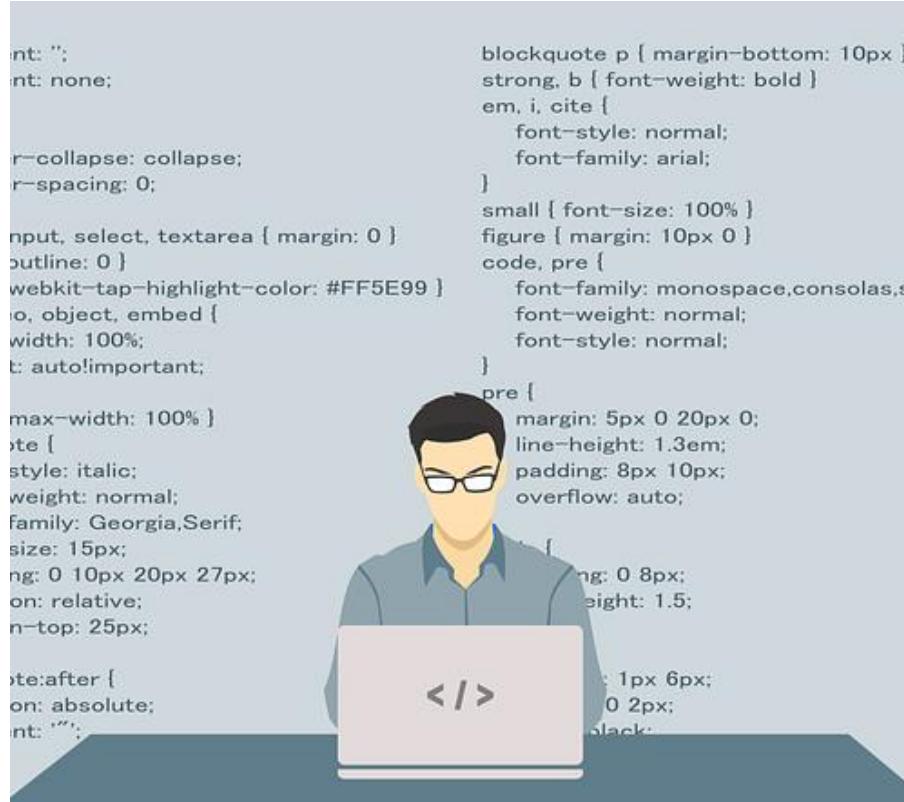
- “Theoretical” overview of the ML pipeline steps
  - Data Gathering
  - Feature engineering & selection
  - Model building & assessment



## Second half

- Practical implementation of all of the above
- **With code**

# • Sections 5 onward: Production Code



- Package the model
- Create a model API
- Deploy to PaaS
- Testing
- More...
- **Hands on model deployment - The gist of the course**





# What does this mean for you?

Where should you start?

- Depending on your level and interest

I am at early stages of my career

Follow the course in order

I want to code straightaway

Start from the 2<sup>nd</sup> half of section 4

I am an experienced data scientist

Skip section 4

I am an avid programmer but haven't done ML before

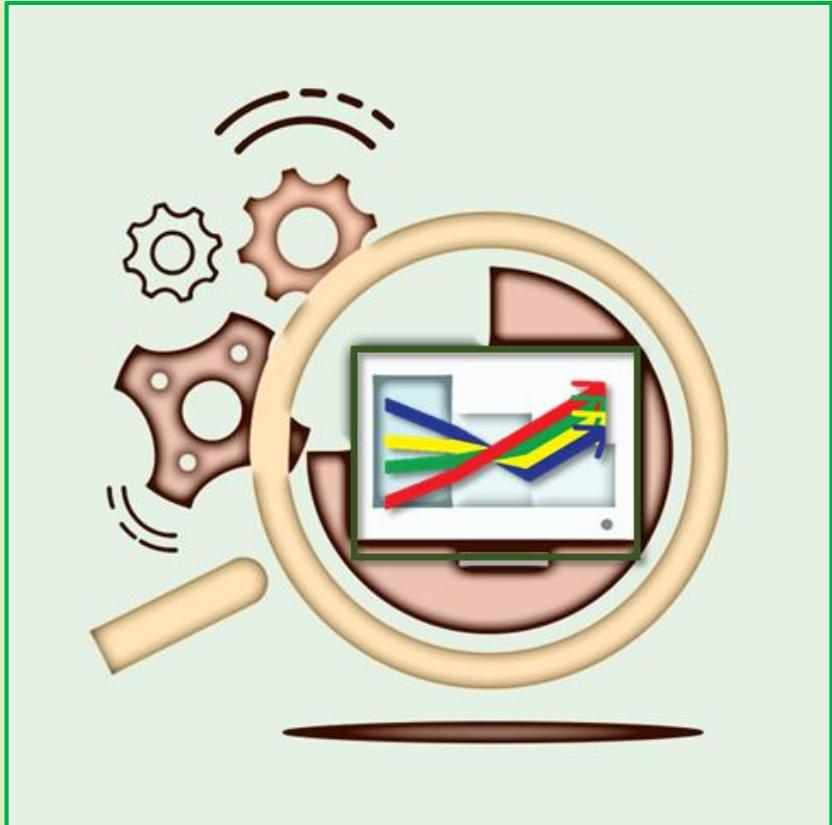
Section 4 gives overview of the process



# THANK YOU

[www.trainindata.com](http://www.trainindata.com)

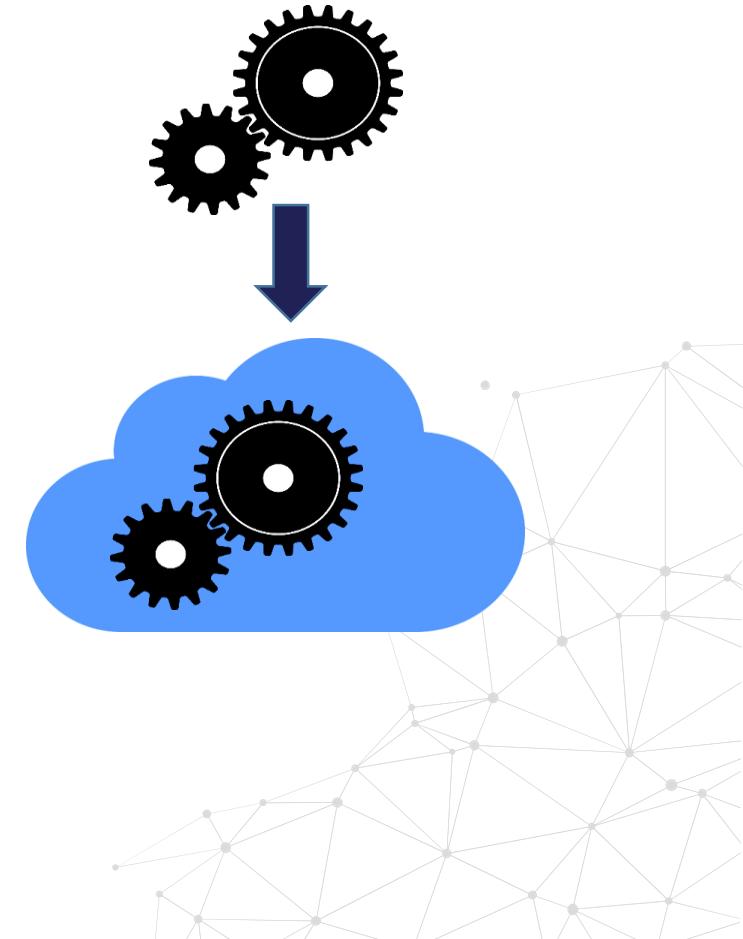




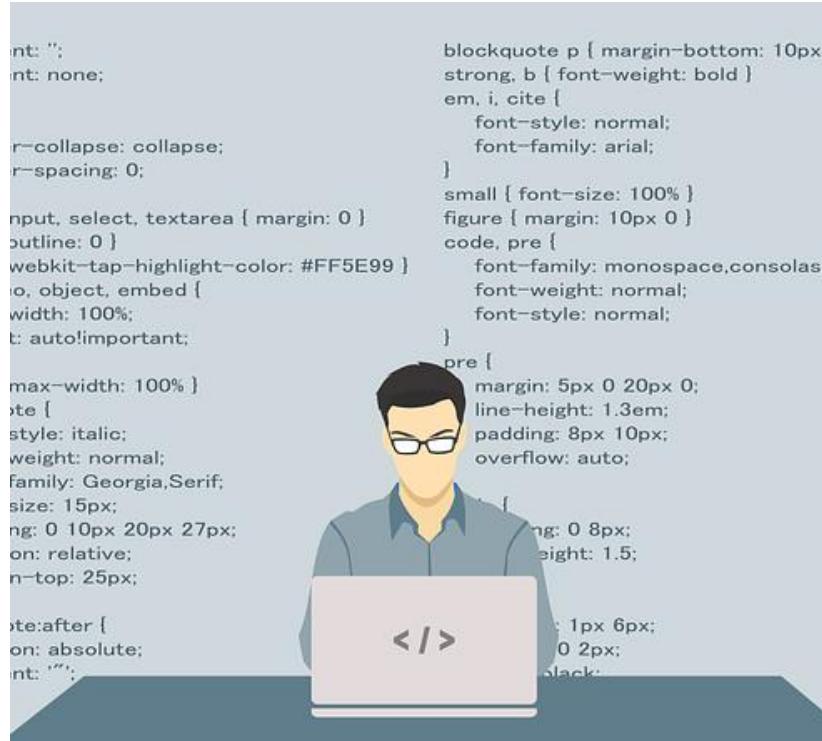
# Deployment of Machine Learning Models

# • What is Model Deployment?

- The deployment of machine learning models is the process for making models available in production environments, where they can provide predictions to other software systems.
- One of the last stages in the Machine Learning Lifecycle.
- Potentially the most challenging stage.



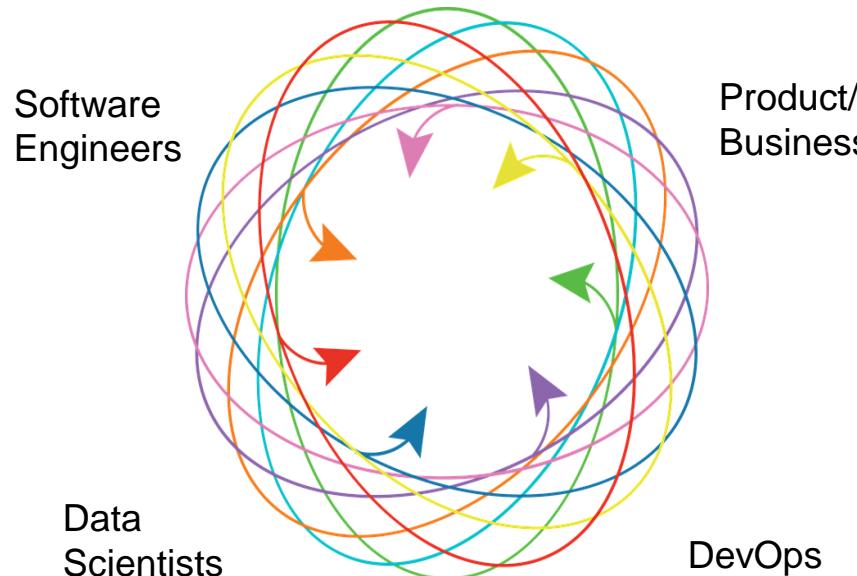
# • Why is Model Deployment Challenging?



- Challenges of traditional software
  - Reliability
  - Reusability
  - Maintainability
  - Flexibility
- Additional challenges specific to Machine Learning
  - Reproducibility



# • Why is Model Deployment Challenging?



- Needs coordination of data scientists, IT teams, software developers and business professionals:
  - Ensure model works reliably
  - Ensure model delivers the intended result.
  
- Potential discrepancy between programming language in which the model is developed and the production system language.
  - Re-coding the model extends the project timeline and risks lack of reproducibility



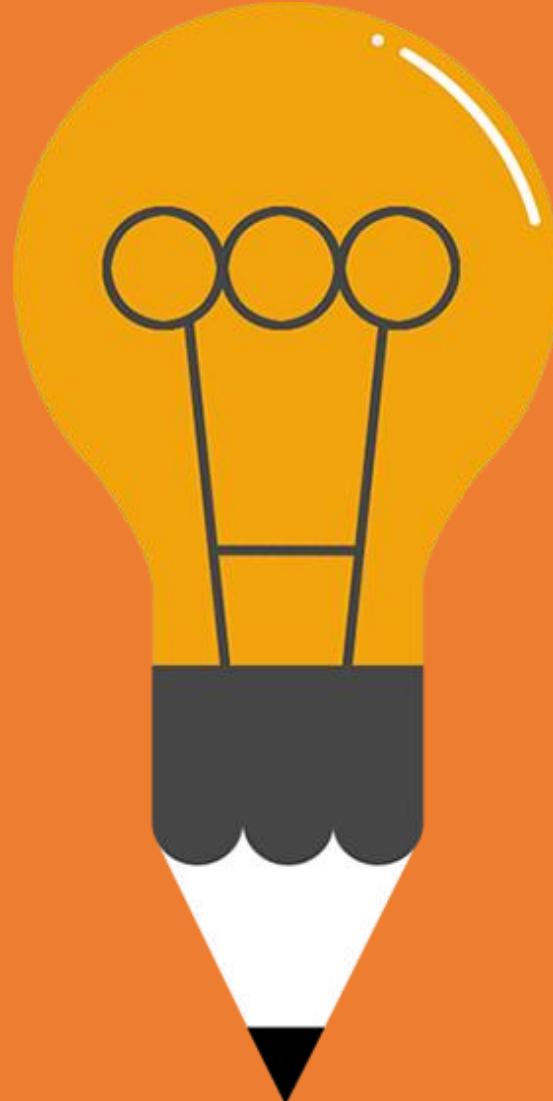
# • Why is Model Deployment important?

- To start using a Machine Learning Model, it needs to be effectively deployed into production, so that they can provide predictions to other software systems.
- To maximize the value of the Machine Learning Model, we need to be able to reliably extract the predictions and share them with other systems.



- In the course...

Discuss **best practices** and **solutions** to mitigate these challenges and streamline the integration of machine learning models in production.

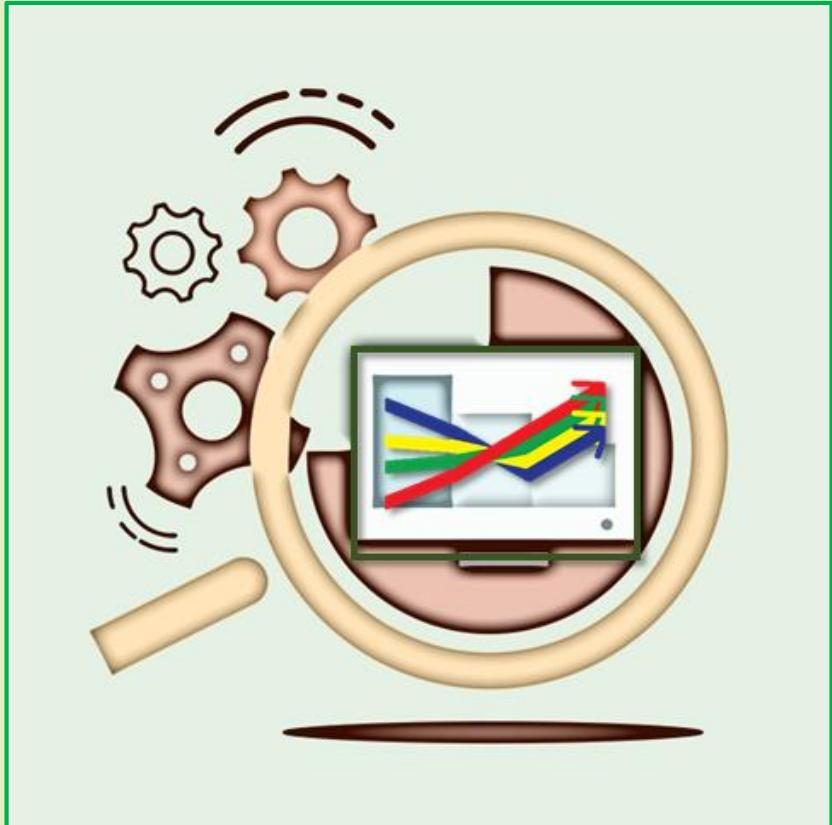




# THANK YOU

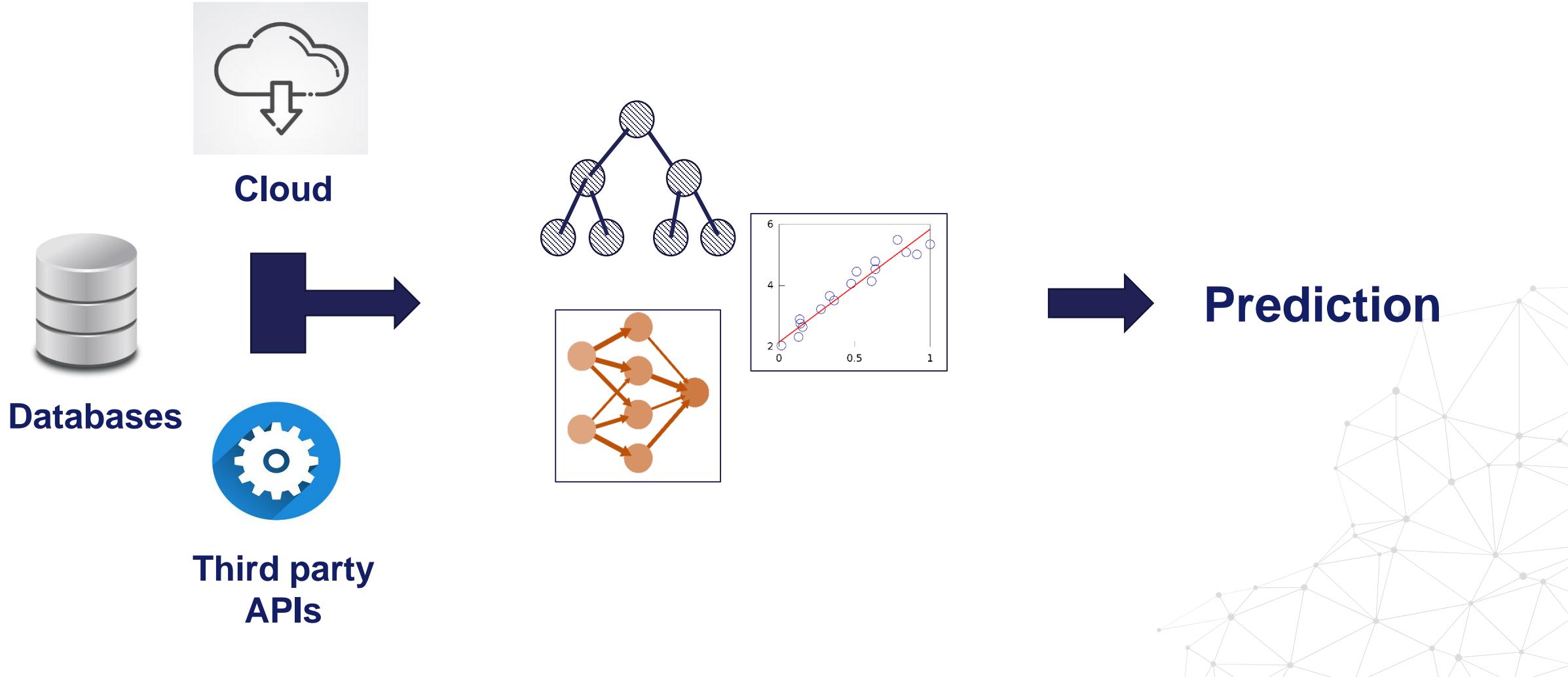
[www.trainindata.com](http://www.trainindata.com)



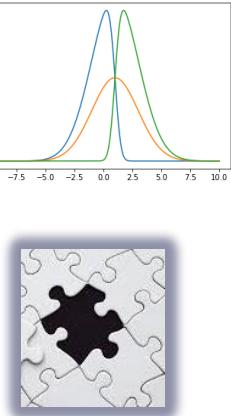
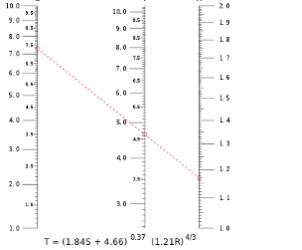


# Deployment of Machine Learning Pipelines

# Machine Learning Models



# Data Format and Quality

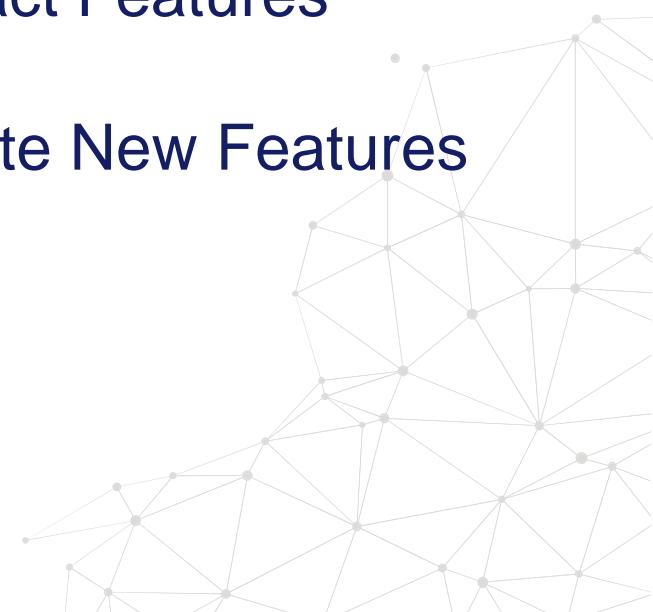


Data

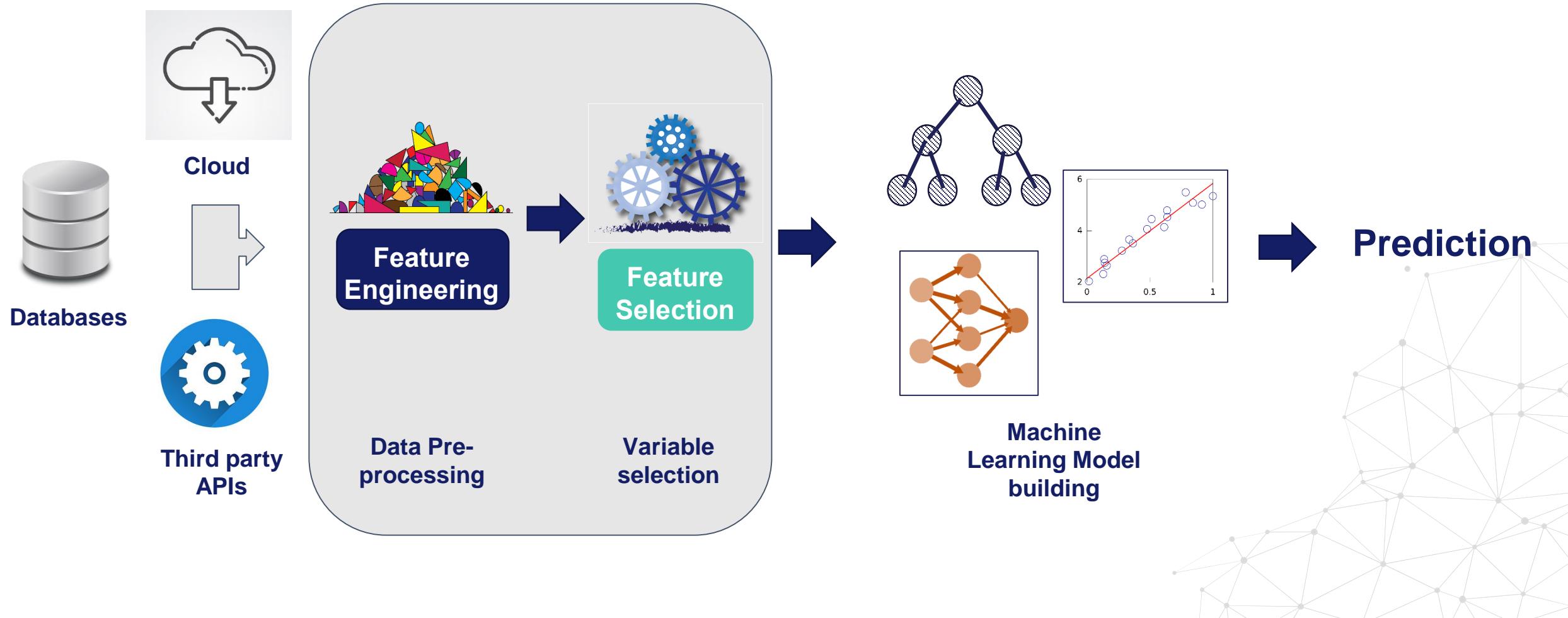
An insurance claim  
A formal request to an insurer for payment based on the terms of an insurance policy.  
Insurance claims are recorded by the insurer and used to calculate premiums.



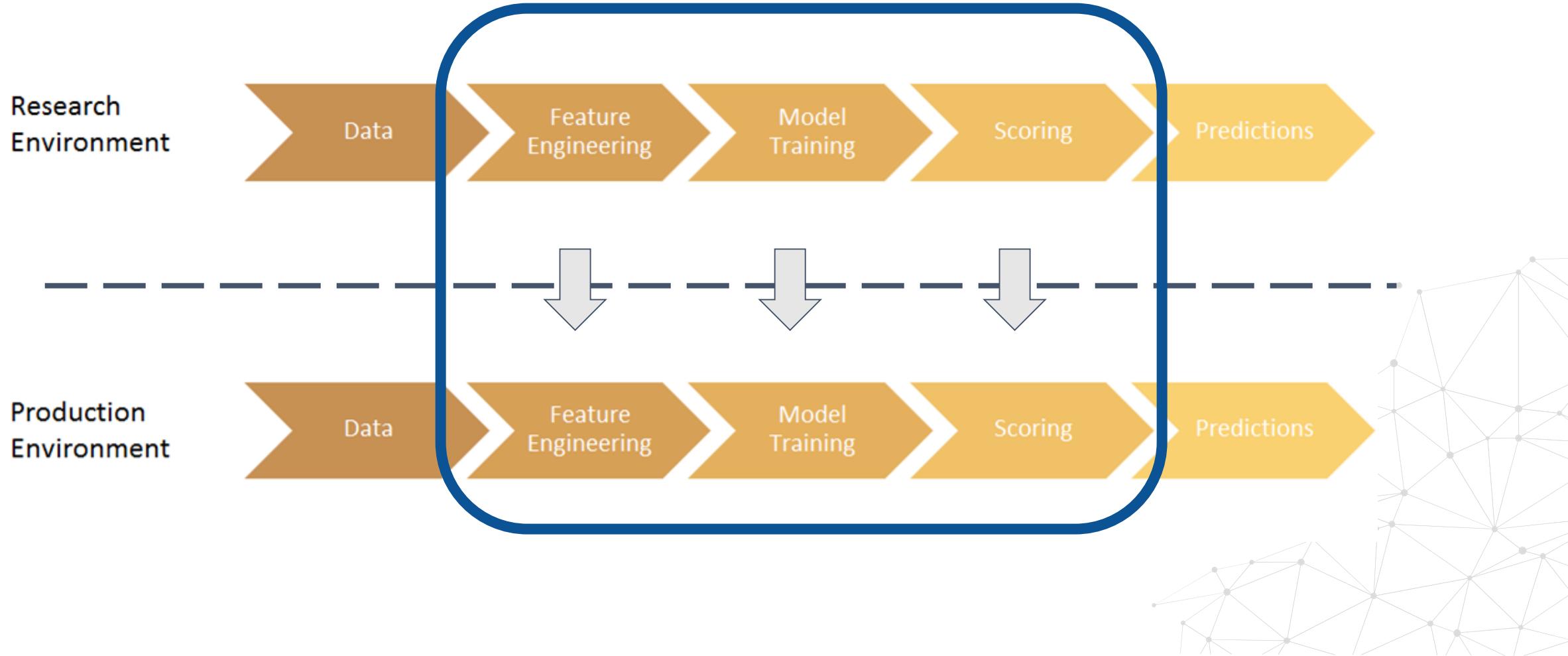
- Transform Variables
- Extract Features
- Create New Features



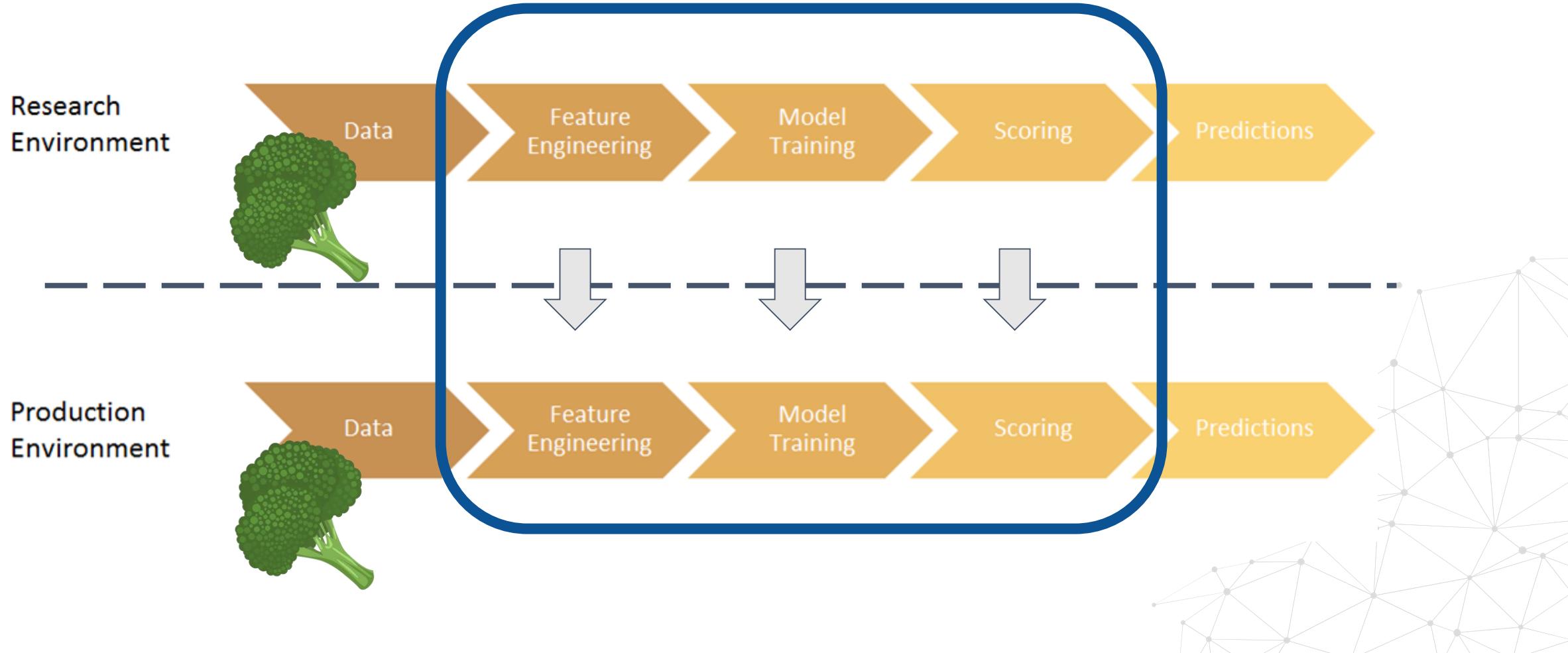
# Machine Learning Pipeline



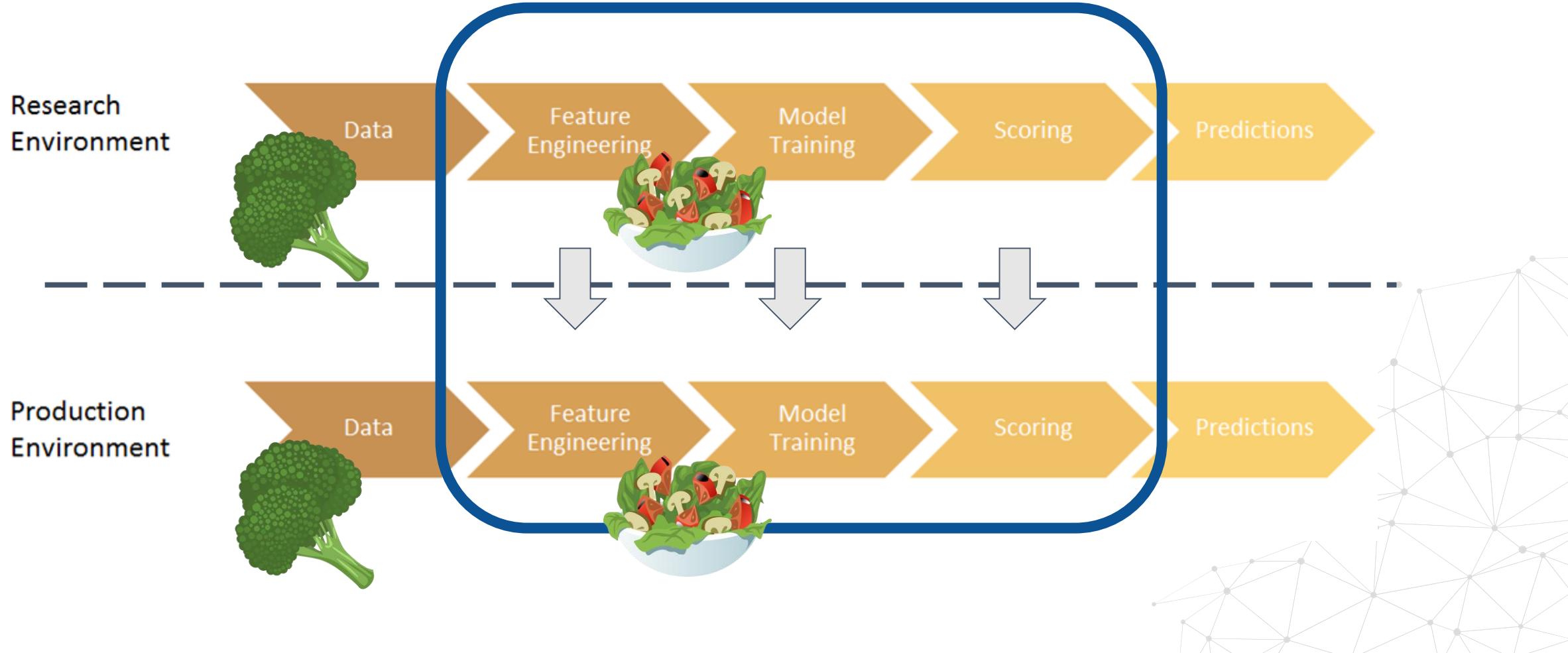
# Deployment of ML Pipeline



# Deployment of ML Pipeline



# Deployment of ML Pipeline

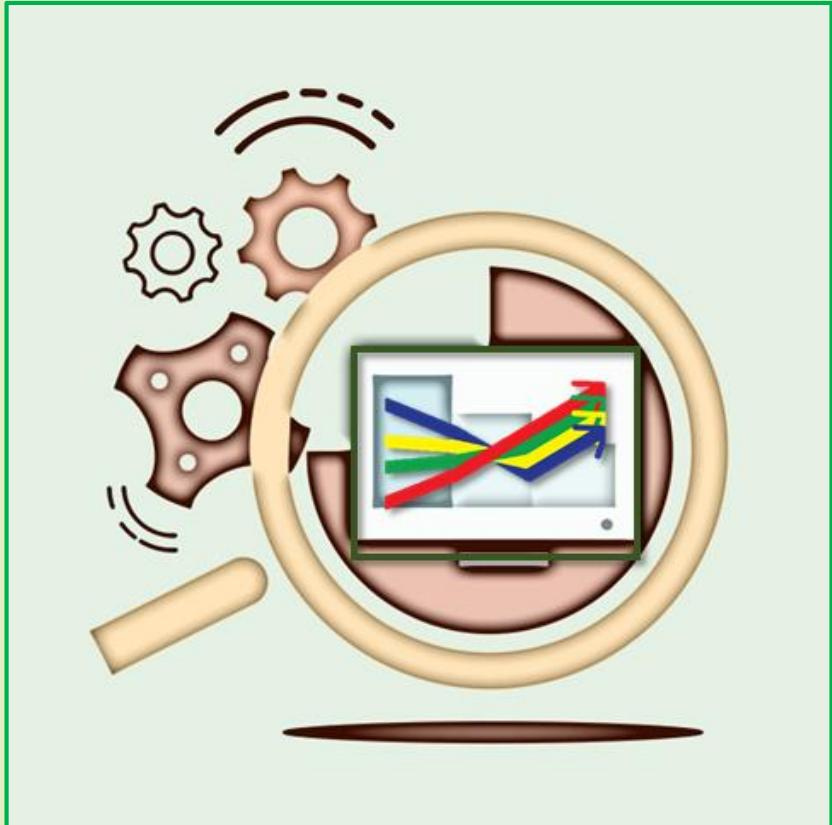




# THANK YOU

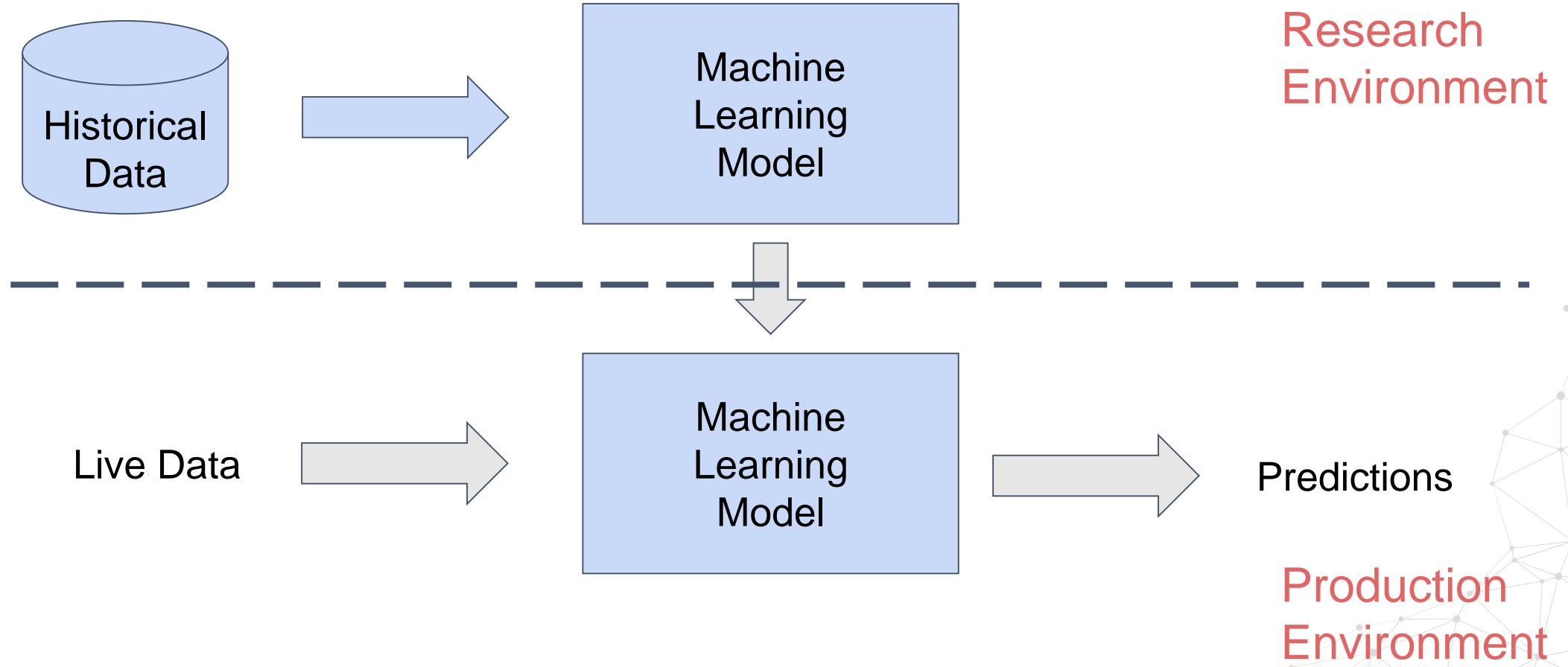
[www.trainindata.com](http://www.trainindata.com)





# Research and Production Environments

# Research and Production Environment



Train In Data

# Environments

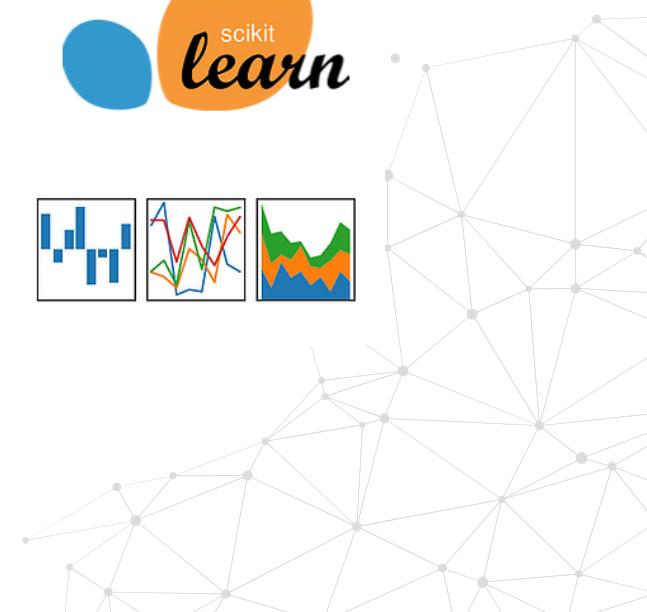
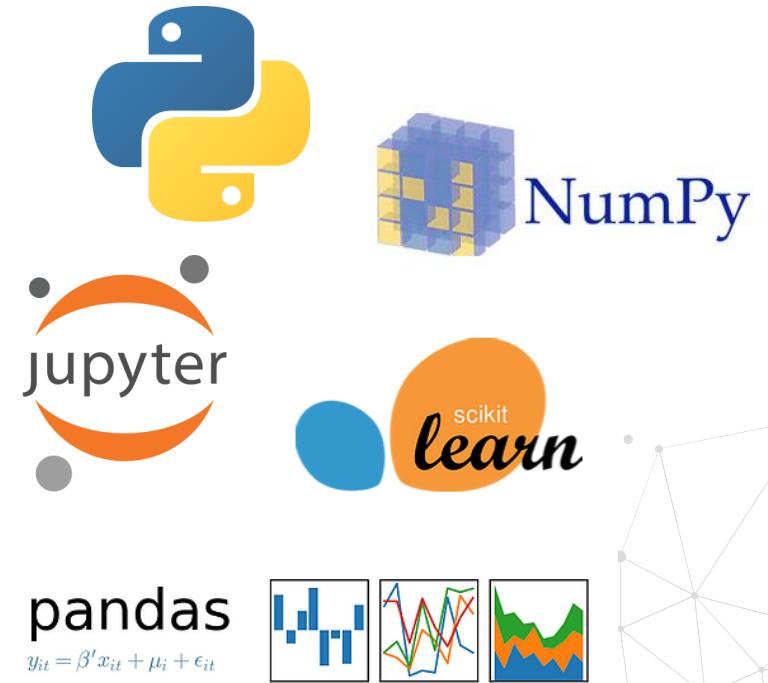
The term **environment** refers to the setting or state of a computer where software or other products are developed or put in operation.

- Software, hardware, programs and operating systems



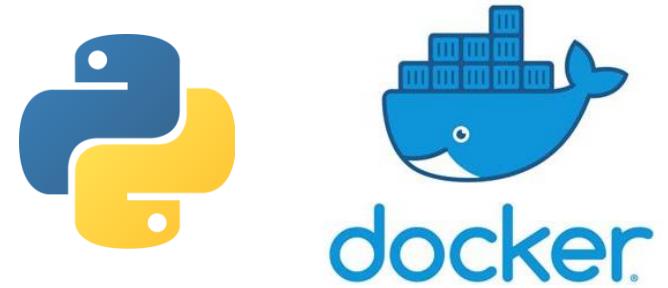
# • Research Environment

- The Research Environment is a setting with tools, programs and software suitable for data analysis and the development of machine learning models.
- Here, we develop the Machine Learning Models and identify their value.



# Production Environment

- The Production Environment is a real-time setting with running programs and hardware setups that allow the organization's daily operations.
- It's the place where the machine learning models is actually available for business use.
- It allows organisations to show clients a “live” service.

A screenshot of the PyCharm 2020.3.3 IDE interface. The left pane shows a project structure for 'Feature Engine' with various Python files like 'decision\_tree.py', 'equal\_frequency.py', and 'equal\_width.py'. The right pane displays the code for 'decision\_tree.py':

```
def __init__(self, scoring='neg_mean_squared_error', variables=None, intv=None, lbtv=None, rbtv=None, random_state=None):  
    self.scoring = scoring  
    self.intv = intv  
    self.lbtv = lbtv  
    self.rbtv = rbtv  
    self.variables = variables  
    self.random_state = random_state  
  
    if param_grid is None:  
        param_grid = {'cv_depth': [1, 2, 3, 4]}  
  
    if not isinstance(cv, int) or cv < 0:  
        raise ValueError("cv can only take non-negative integers")  
  
    if not isinstance(regression, bool):  
        raise ValueError("regression can only take True or False")  
  
    self.cv = cv  
    self.scoring = scoring  
    self.regression = regression
```

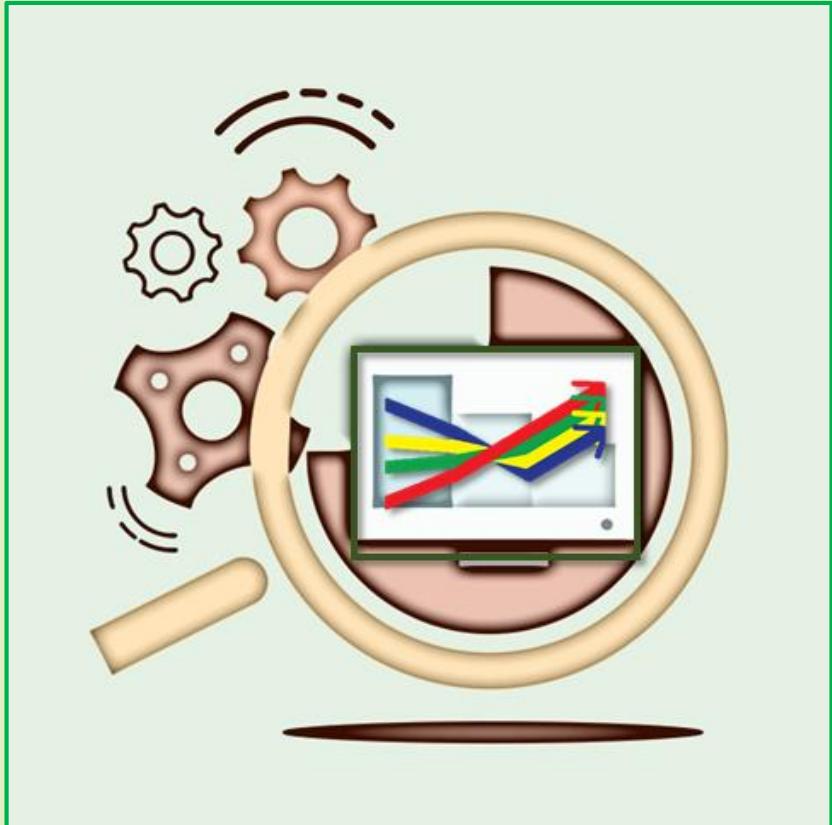
The bottom of the screen shows the Windows taskbar with the PyCharm icon and a message: "PyCharm 2020.3.3 available".



# THANK YOU

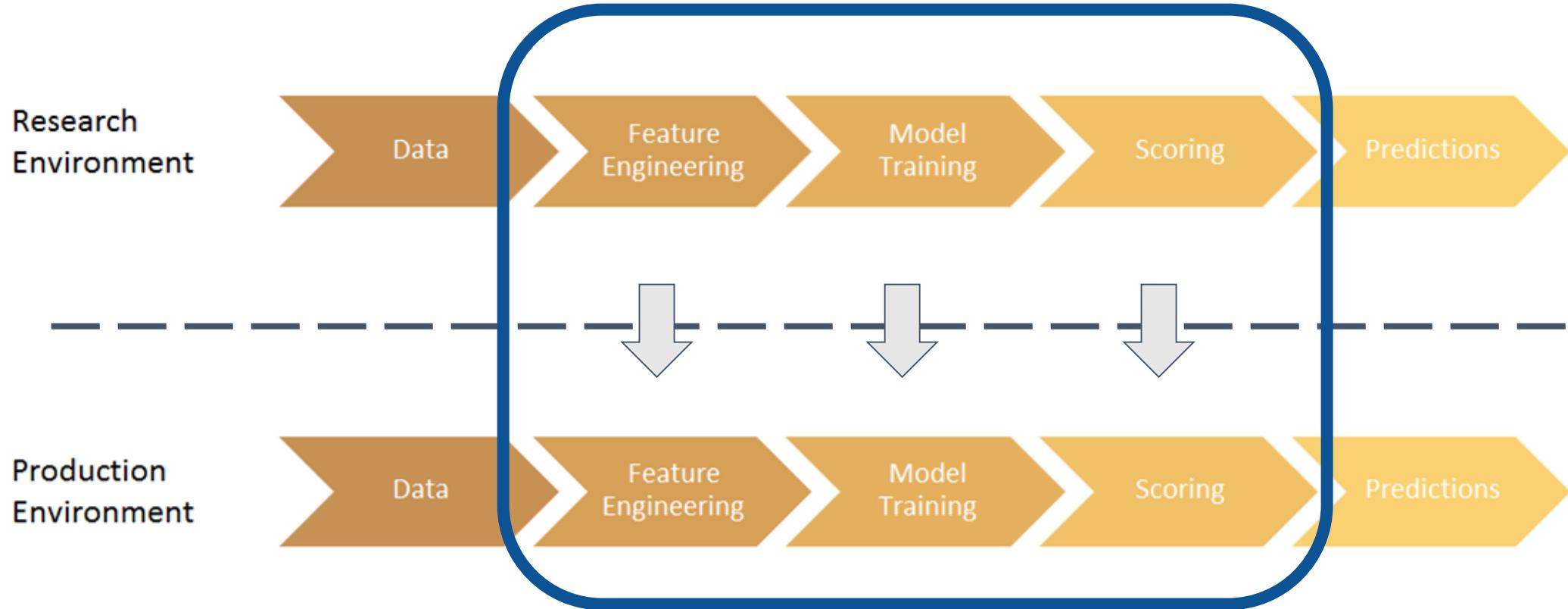
[www.trainindata.com](http://www.trainindata.com)



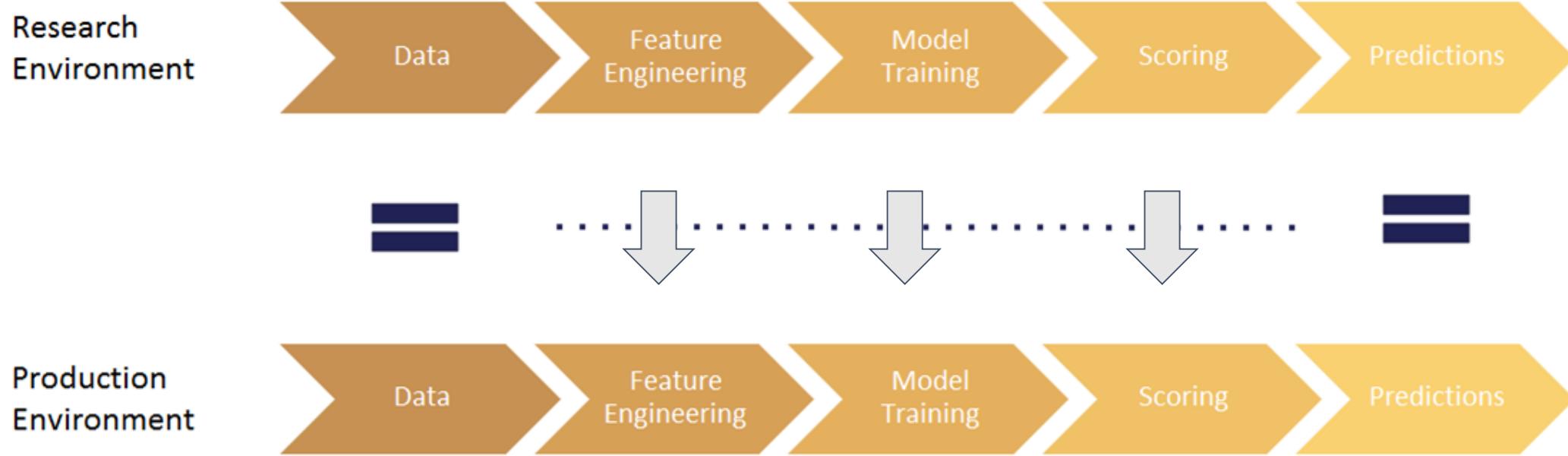


# Building Reproducible Machine Learning Pipelines

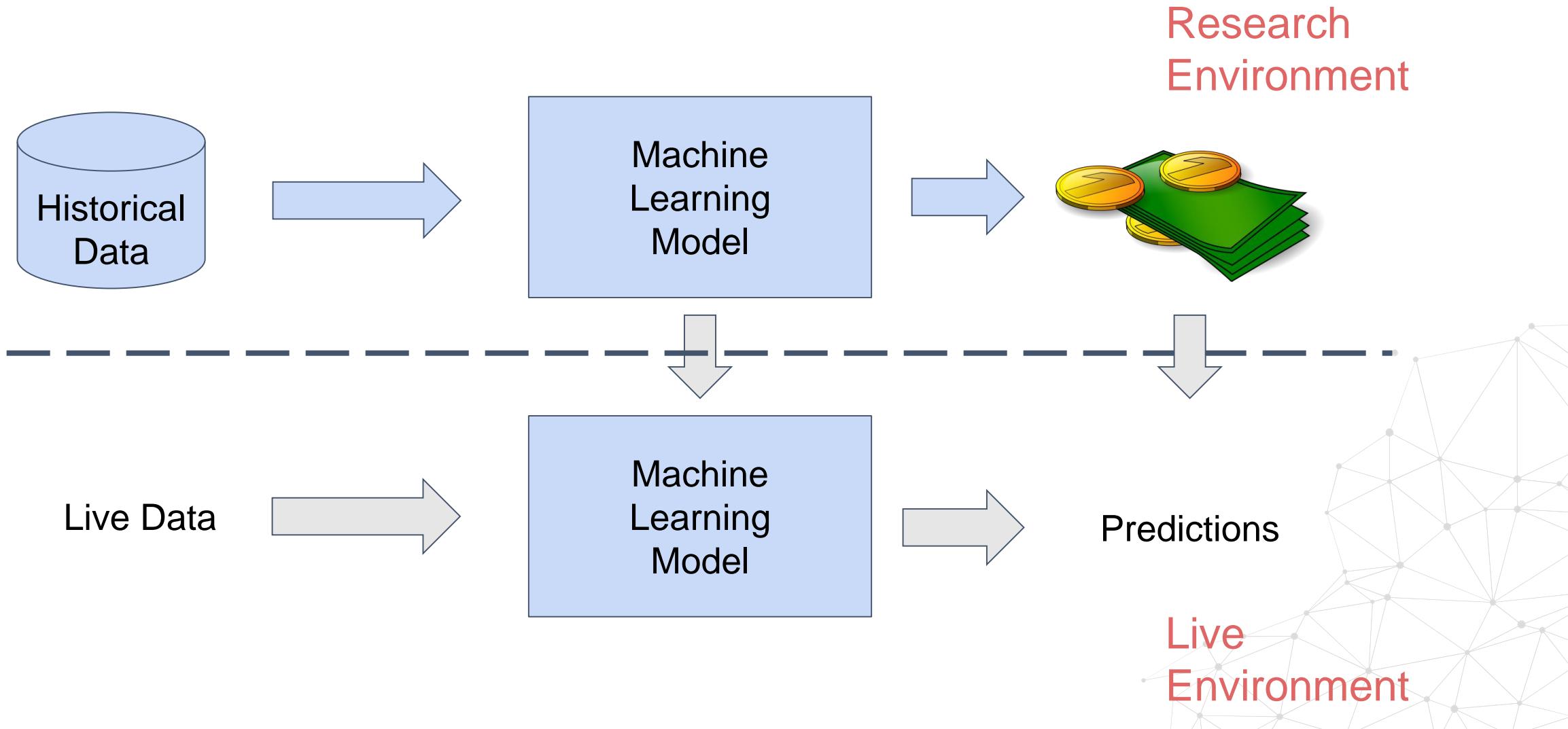
# Deployment of ML pipelines



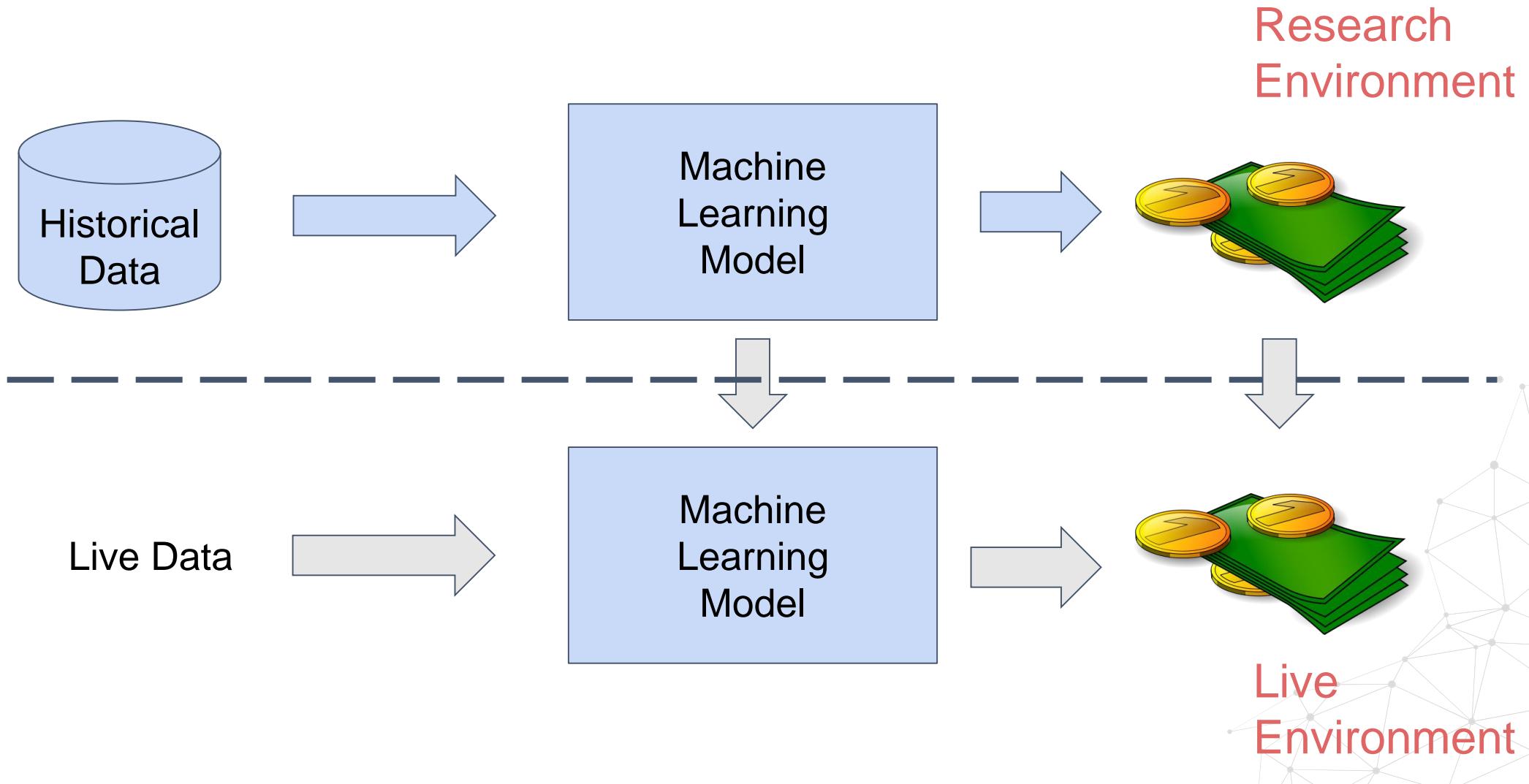
# Reproducible ML pipelines



# Deploying the value



# Deploying the value



# Why Reproducibility Matters?

1. Financial costs
2. Lost time
3. Lost reputation



**Without the ability to replicate prior results, it is difficult to determine if a new model is truly better than the previous one.**



# THANK YOU

[www.trainindata.com](http://www.trainindata.com)

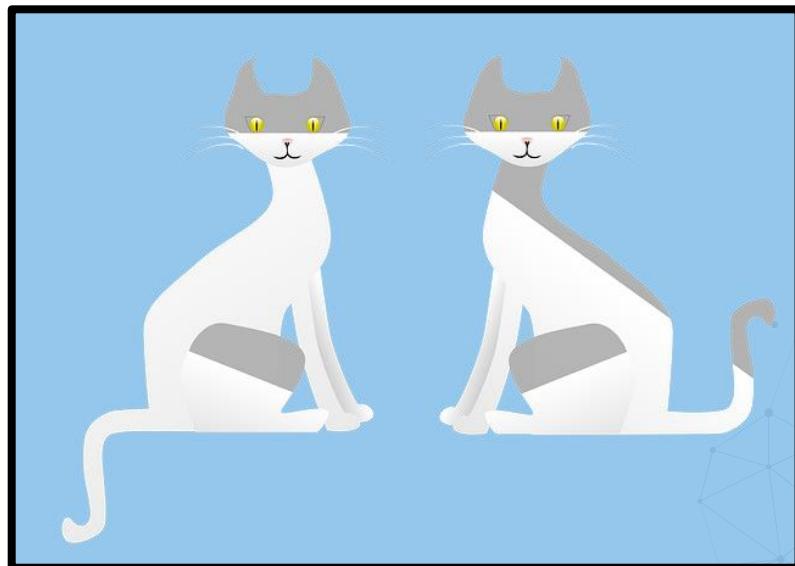




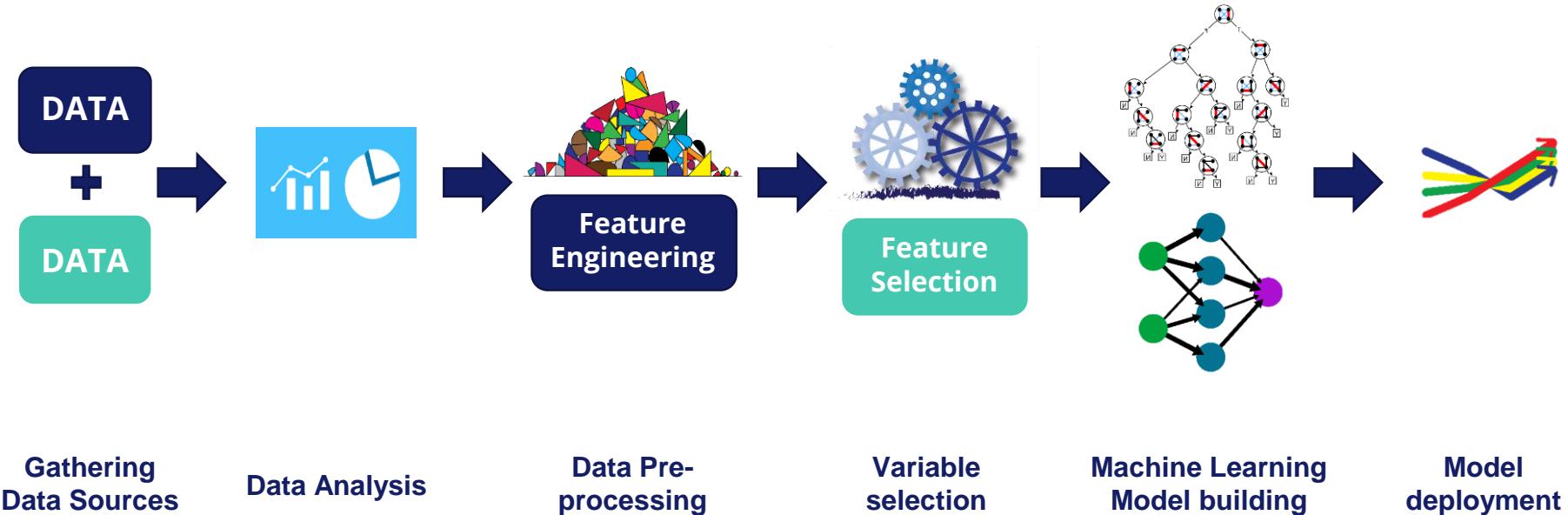
# Challenges to Reproducibility

# Reproducibility in Machine Learning

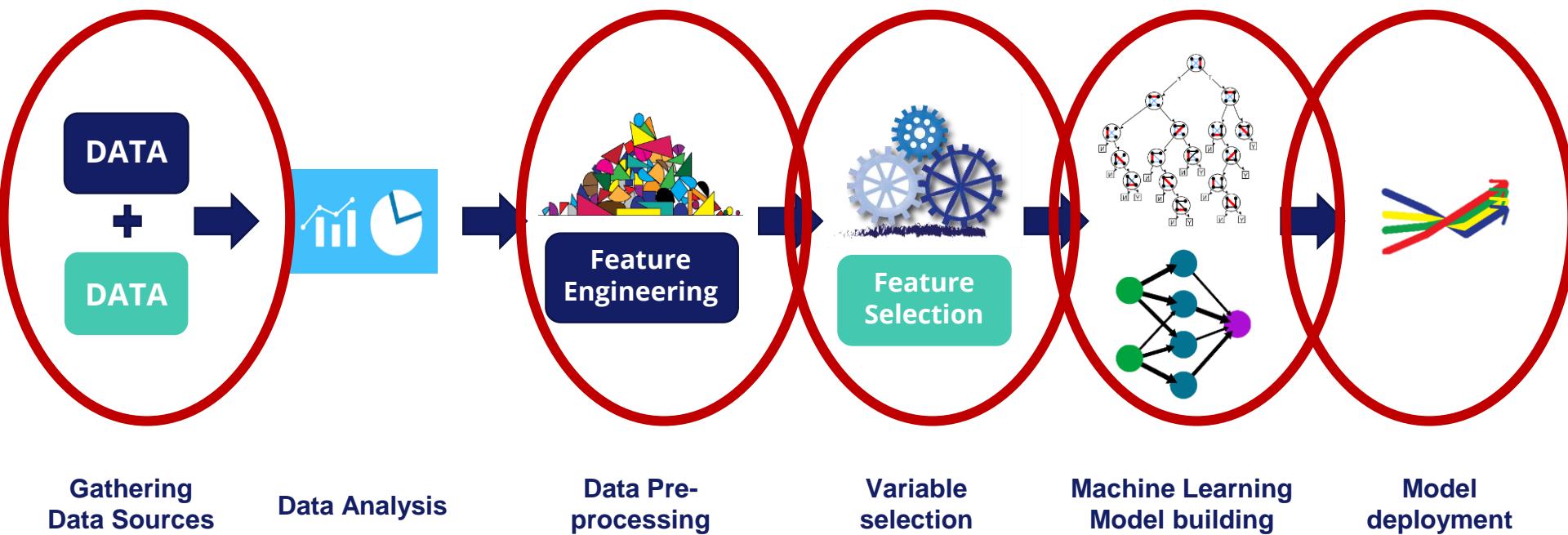
**Reproducibility** is the ability to duplicate a machine learning model exactly, such that given the same raw data as input, both models return the same output.



# Machine Learning Pipeline



# Machine Learning Pipeline



# Reproducibility during Data Gathering

Data → The most difficult challenge to reproducibility



Cloud



Databases



## Challenges

- ❖ Training dataset can't be reproduced
- Databases are constantly updated and overwritten.
- Order of data while loading is random (SQL).



Third party APIs

## Solutions

- ❖ Save a snapshot of training data
  - ✓ Simple
    - Potential conflict with GDPR
    - Not suitable for big data
- ❖ Design data sources with accurate timestamps.
  - ✓ Ideal situation
    - Big effort to (re)design the data sources

# Reproducibility during Feature Creation



Feature  
Engineering

Data Pre-  
processing

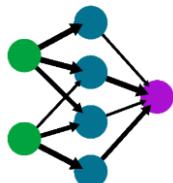
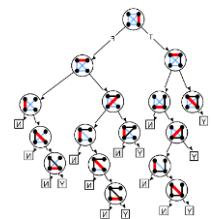
## Challenges

- ❖ Replacing missing data with random extracted values
- ❖ Removing labels based on percentages of observations
- ❖ Calculating statistical values like the mean to use for missing value replacement
- ❖ More complex equations to extract features, e.g., aggregating over time

## Solutions

- ❖ Code on how a feature is generated should be tracked under version control.
- ❖ Many of the parameters extracted for feature engineering depend on the data used for training → ensure data is **reproducible**
- ❖ If replacing by extracting random samples, always set a seed

# Reproducibility during Model Training



## Challenges

- ❖ Machine learning models rely on randomness for training
    - Data and feature extraction for trees
    - Weight initialisation for neural nets, etc.
  - ❖ Machine Learning model implementations work with arrays agnostic to feature names
    - Need to be careful to feed data in the correct order

## Solutions

- ❖ Record the order of the features
  - ❖ Record applied feature transformations
  - ❖ Record hyperparameters
  - ❖ For models that require randomness always set a seed.
  - ❖ If the final model is a stack of models, record the structure of the ensemble.

# Reproducibility during Model Deployment:



## Challenges

- ❖ A feature is not available in the live environment
- ❖ Different programming languages
- ❖ Different software
- ❖ Live populations don't match those used for training

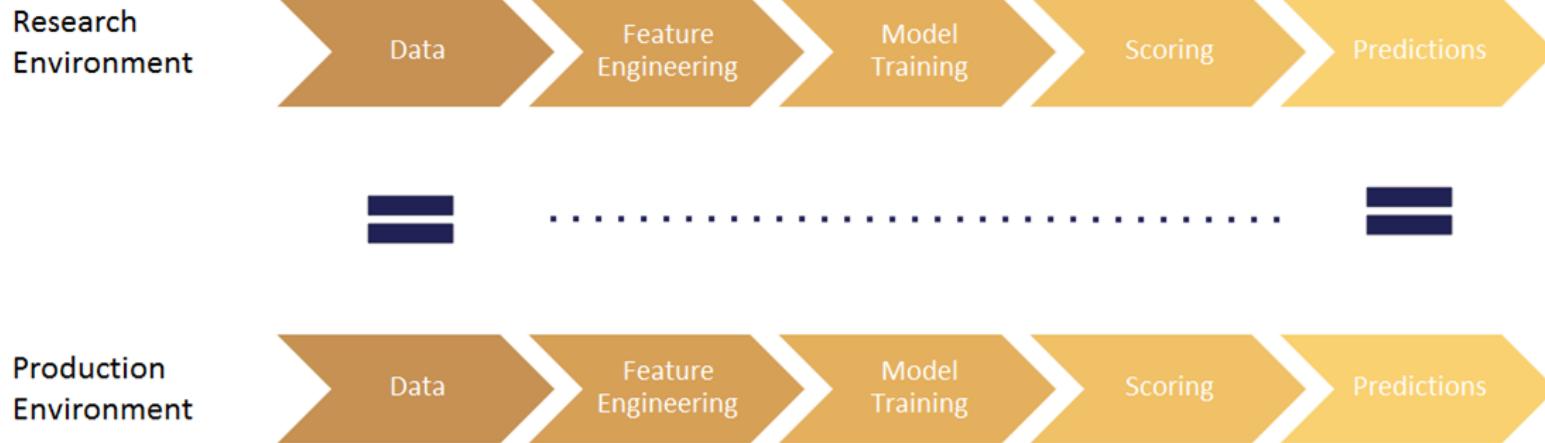
## Solutions

- ❖ Software versions should match exactly - applications should list all third party library dependencies and their versions
- ❖ Use a container and track software specifications
- ❖ Research, develop and deploy utilising the same language, e.g., python
- ❖ Prior to building the model, understand how the model will be integrated with other systems



# Streamlining Model Deployment with Open-Source

# Reproducible ML Pipelines



# Machine Learning Pipeline



- Missing data imputation
- Categorical encoding
- Discretisation
- Transformation
- Feature extraction
  - Text
  - Datetime
  - Images
  - Time series
- Feature combination
- **Lots to code!**
- Linear Models
- Decision trees
- KNNs
- SVMs
- Neural Networks
- Stacking
- **Lots of parameters → Reproducibility**

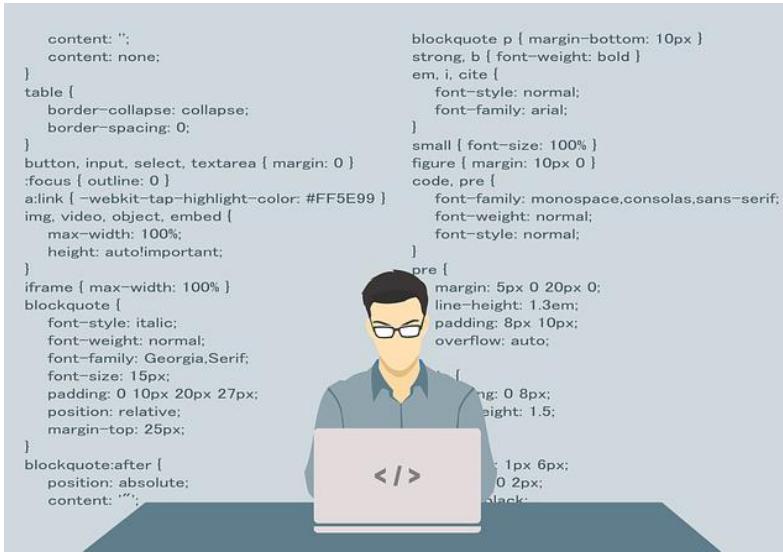


# Challenges

- A lot to code
- Repetitive
- Learn and store parameters
- Reproducibility



# Lots to code



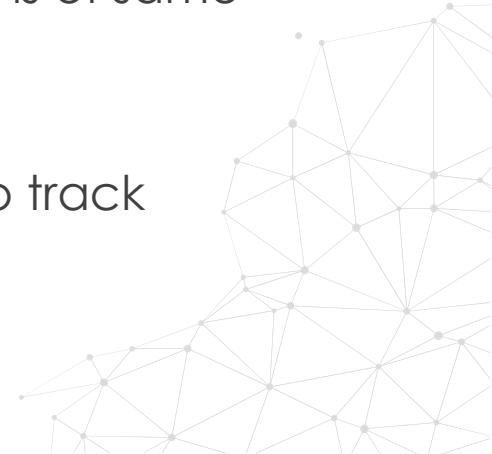
- Time consuming
- Different versions across team



# Repetitive



- Multiple copies of same code
- Different versions of same code
- Difficult to keep track



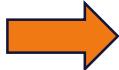
# Learn and store parameters



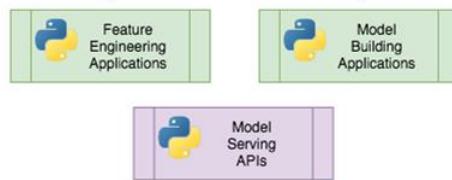
- Multiple intermediate files with parameters
- Config or params file



# Reproducibility in Deployment



Research  
Environment

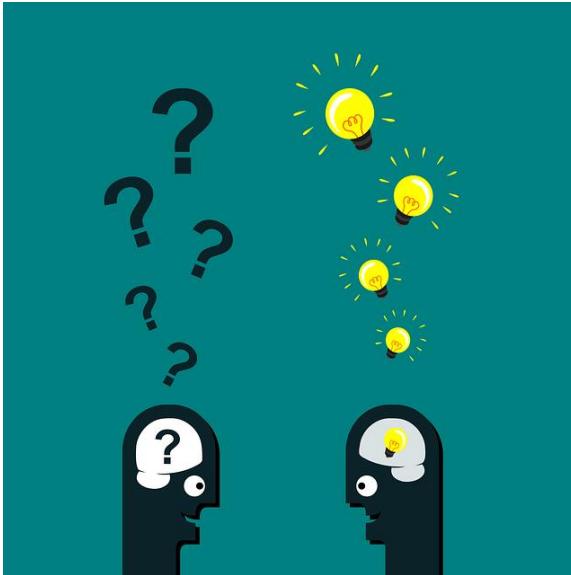


Production  
Environment

- Re-write code
- Include tests
- Reproducibility



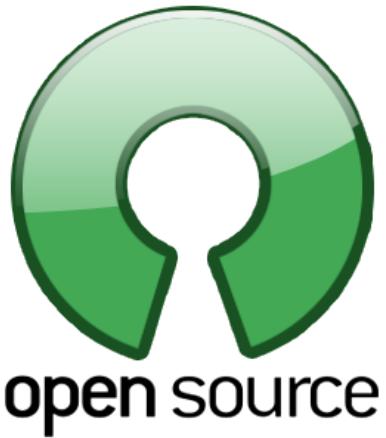
# Team Performance



- Decreased Performance
- Frustration
- Lack of reproducibility
- Increased deployment times



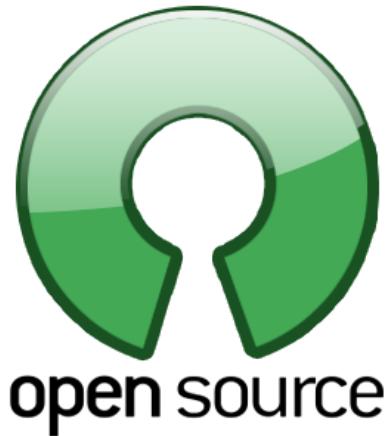
# Open-source



- Increase Performance
- Prevent Frustration
- Maximise reproducibility
- Minimise deployment times



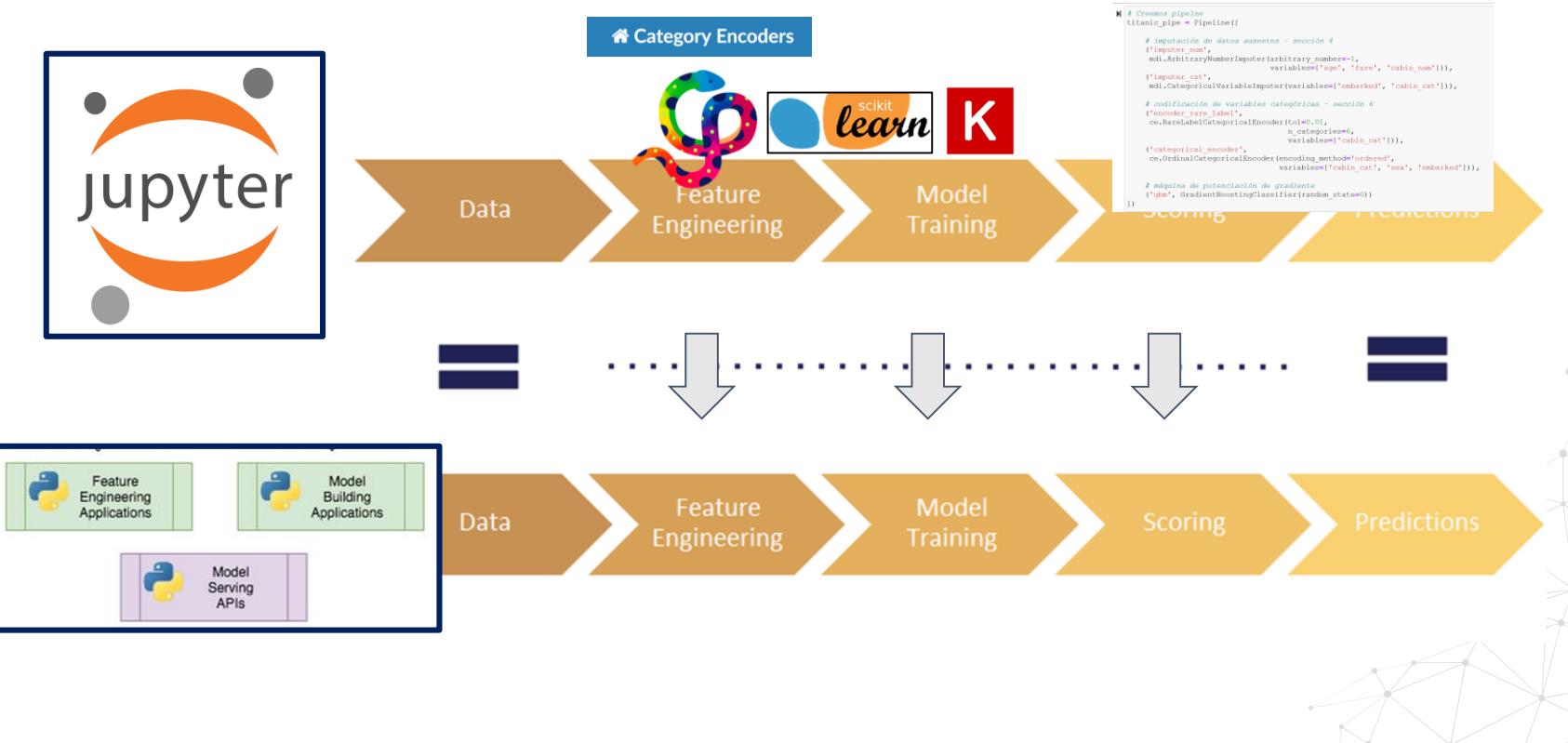
# Open-source



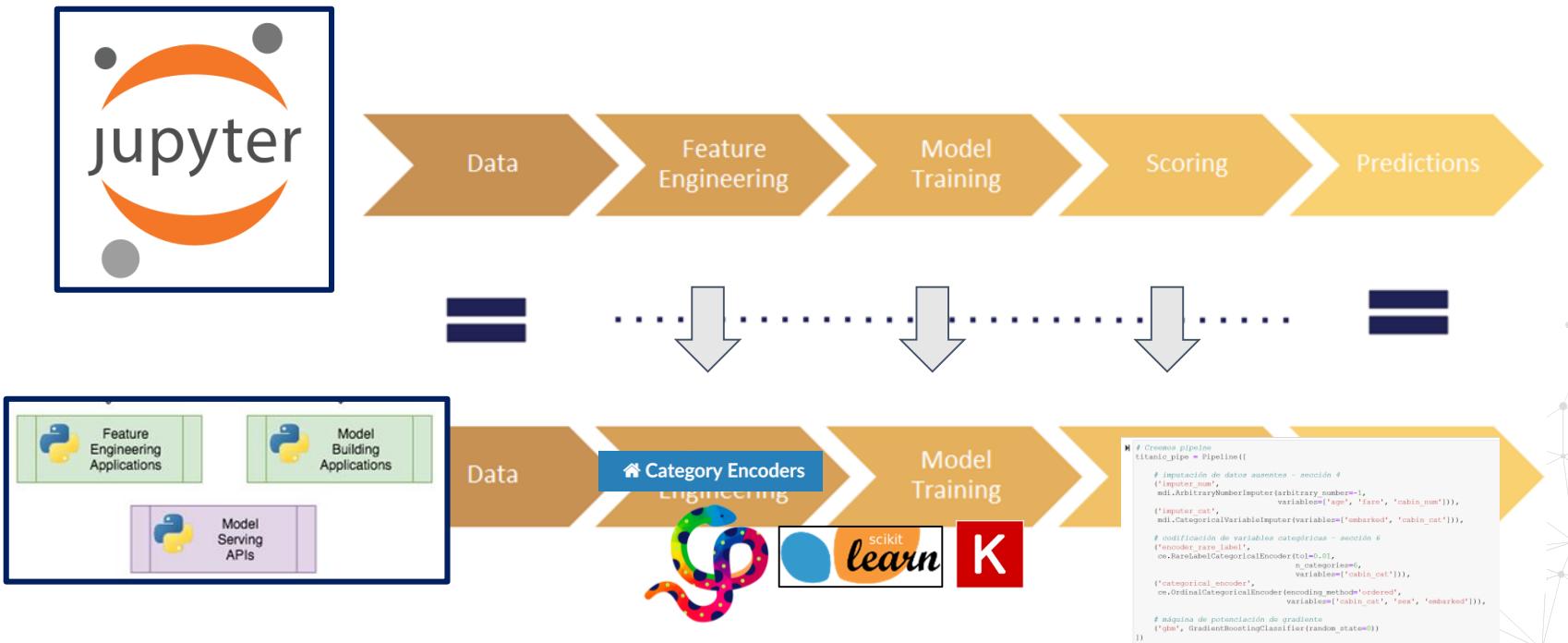
- No more coding
- Version tracking for reproducibility
- Classes and functions include tests – no need to recode for production



# Reproducible ML Pipelines



# Reproducible ML Pipelines





# Machine Learning Architecture

(and why it matters)

# What do we mean "System Architecture"?



- Let's dig into the terminology to avoid confusion...





# Systems

Machine learning in production requires multiple different components in order to work:

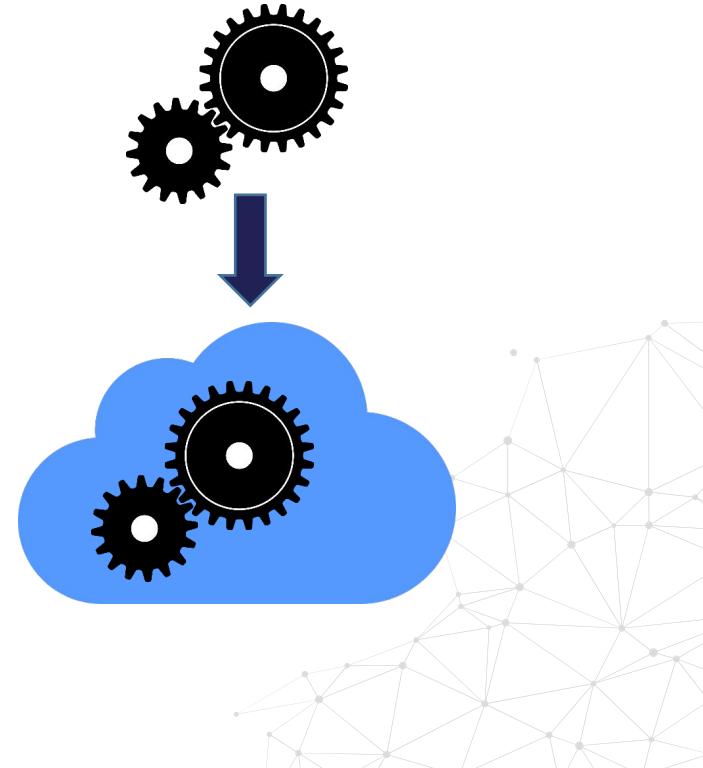
- Infrastructure
- Applications
- Data
- Documentation
- Configuration

# Architecture

ISO/IEC 42010 defines **Architecture** as:

“fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”

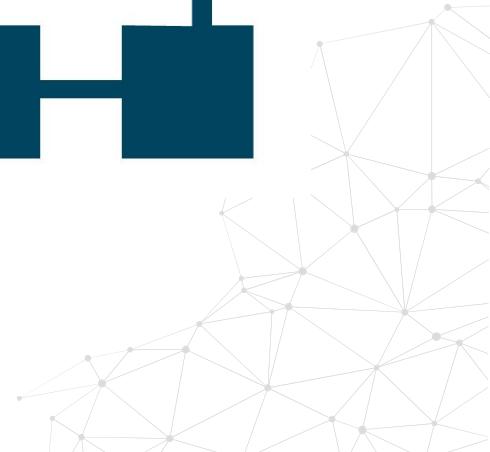
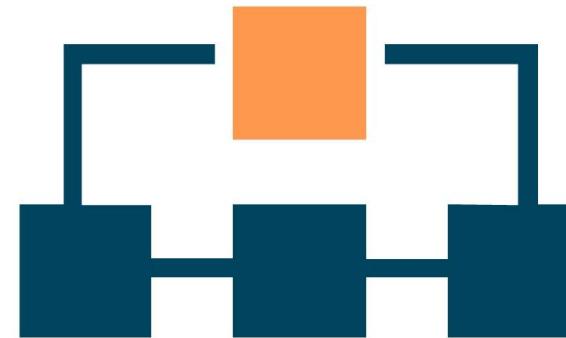
Or, in plain English: “The way software components are arranged and the interactions between them.”



# • Why Start With Architecture - Challenges

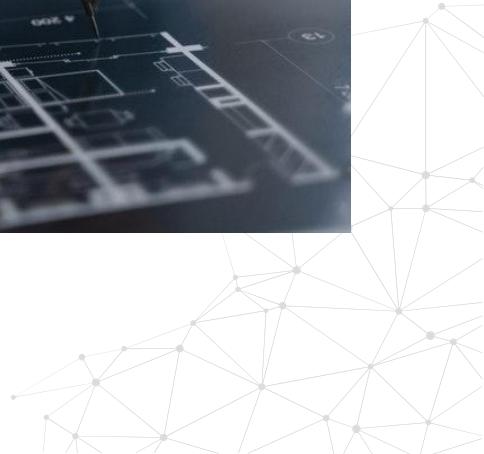
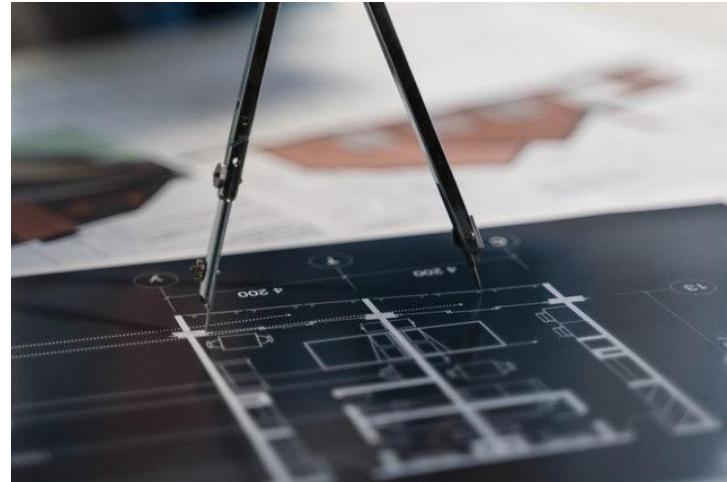
Maintaining ML systems is difficult

All the challenges of traditional software systems *\*plus\** new challenges for model and data changes.



# Section 3 Structure

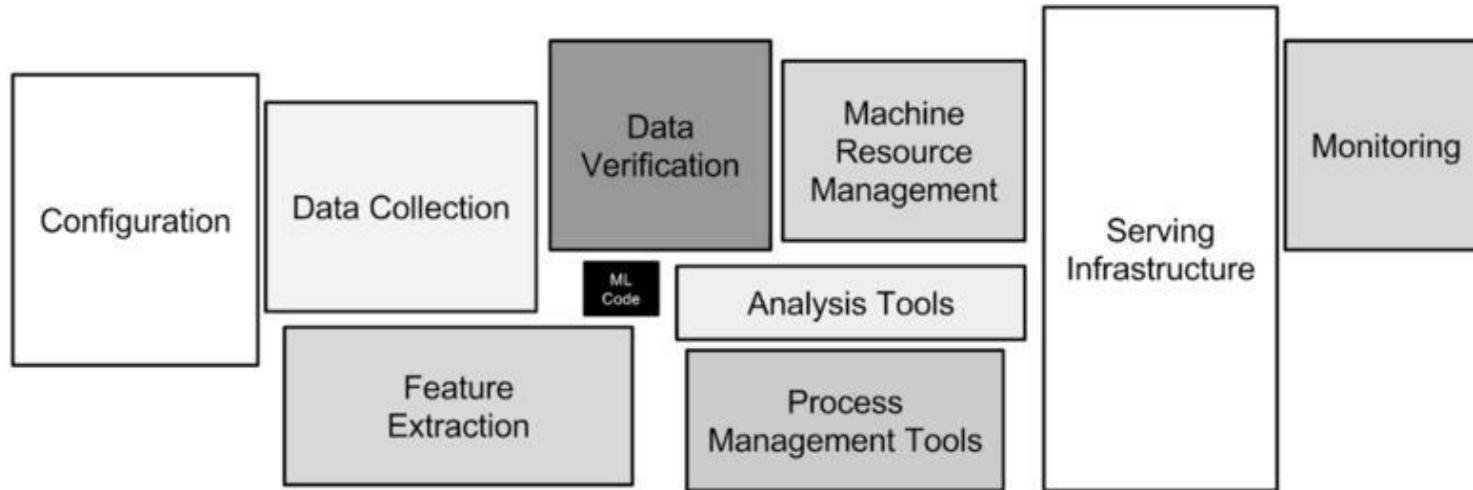
- Lecture 2: Challenges of ML Systems
- Lecture 3: Tackling the Challenges
- Lecture 4: Architecture options
- Lecture 5: Architecture component breakdown





# Challenges of ML Systems

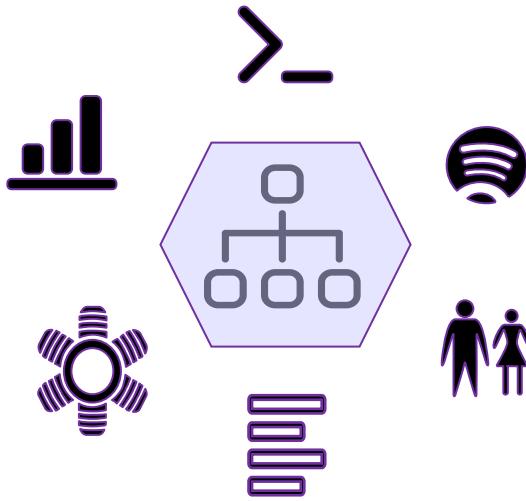
# ML Systems are Complex



Sculley et al. (2014)

<https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf>

# Challenges



- The need for reproducibility  
(versioning everywhere)



# Data Dependencies

Models may be trained on data from many different sources

e.g. a house price prediction model which takes data from:

- An in-house SQL database with information on recent inquiries
- A second in-house NoSQL data store which contains historical house listings
- An external API with the latest crime statistics
- A base-line of features CSV prepared by a data scientist and updated on a weekly basis



# Configuration issues

Model hyperparameters, versions, requirements, data sources can all be changed and modified via config.

e.g. a yaml file in your source code. Is this tested?

```
73  # set train/test split
74  test_size: 0.1
75
76  # to set the random seed
77  random_state: 0
78
79  # The number of boosting stages to perform
80  n_estimators: 50
81
82  # the minimum frequency a label should have to be considered frequent
83  # and not be removed.
84  rare_label_tol: 0.01
85
86  # the minimum number of categories a variable should have in order for
87  # the encoder to find frequent labels
88  rare_label_n_categories: 5
89
90  # loss function to be optimized
91  loss: ls
92  allowed_loss_functions:
93    - ls
94    - huber
```

# Data and Feature Preparation

The steps required to prepare data and transform it into features for the model may be complex.

e.g. a typical pipeline requires us to:

- Transform numerical data
- Transform categorical data
- Handle outliers
- Derive features from raw data
- Many other tasks



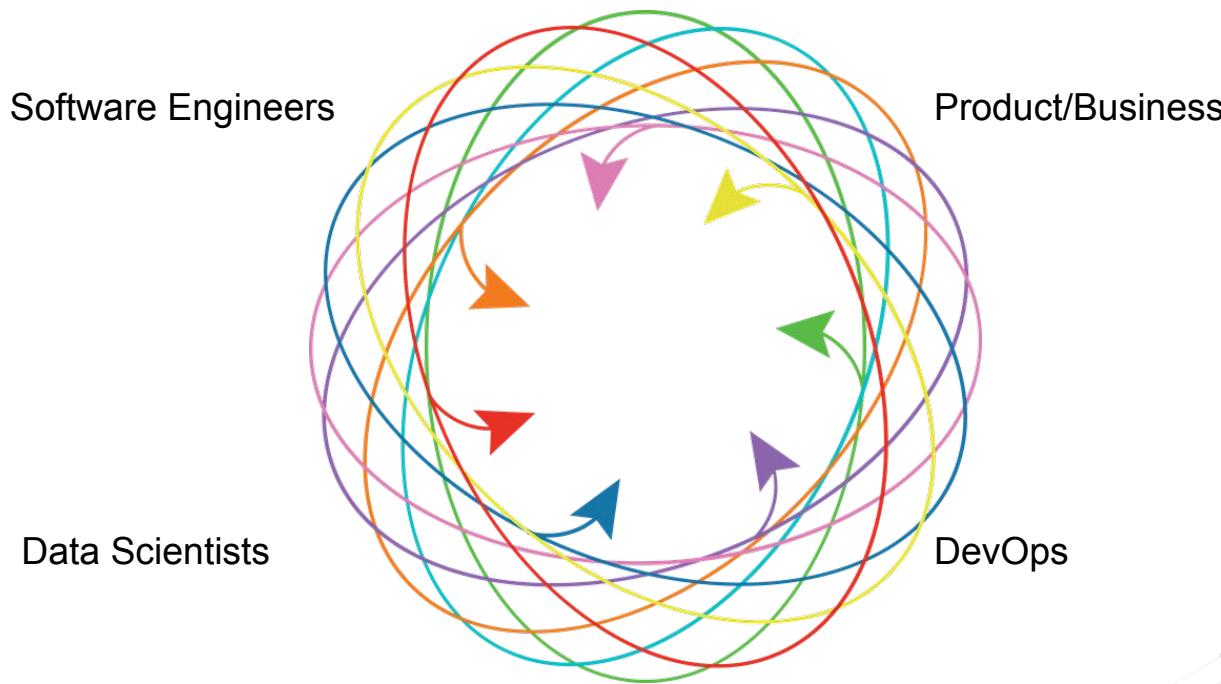
# Detecting Model Errors

Traditional tests often do not detect errors in ML systems

When you deploy a model which performs worse, no exceptions are raised. Your API will not return any 500 status codes. Standard tests will not catch these sorts of mistakes.



# ML System Contributors



# Research vs. Production Environments

	Research	Production
Separate from customer facing software	✓	x
Reproducibility matters	Sometimes	Almost always
Scaling challenges	x	✓
Can be taken offline	✓	x
Infrastructure planning required	Sometimes	Almost always
Difficult to run experiments	x	✓



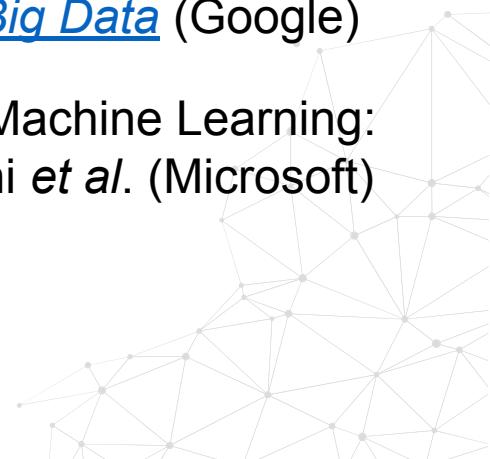


# Key Principles for ML Systems

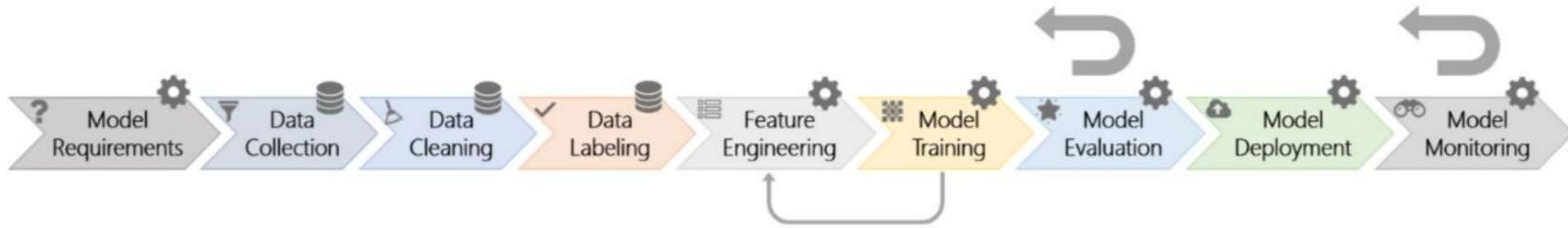
# Useful Resources



1. "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" (2017) Breck *et al.* [IEEE International Conference on Big Data](#) (Google)
2. "Software Engineering for Machine Learning: A Case Study" (2019) Amershi *et al.* (Microsoft)



# End-to-End Pipelines



- Automation of all stages of the ML workflow



# Reproducibility

- Training is reproducible
- Every model specification undergoes a code review and is checked into a repository
- Models can be quickly and safely rolled back to a previous serving version



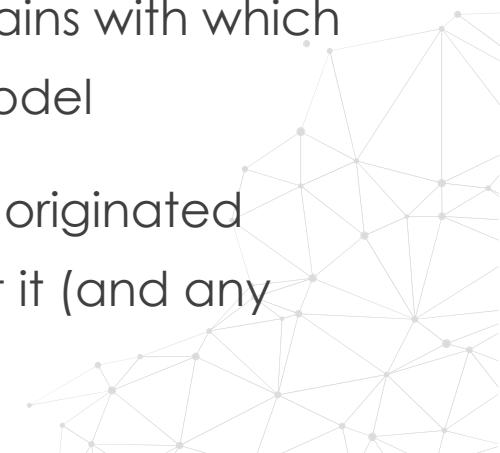
# Versioning

1.2.3-beta.1+meta

The diagram illustrates the structure of a semantic version string. It shows the string "1.2.3-beta.1+meta" with vertical lines extending downwards from each component. The first three components ("1", "2", "3") are grouped under the label "Major". The next two components ("beta", ".1") are grouped under the label "Pre-release". The final component "+meta" is grouped under the label "Metadata".

Major    Minor    Patch    Pre-release    Metadata

- Each model is tagged with a provenance tag that explains with which data it has been trained on and which version of the model
- Each dataset is tagged with information about where it originated from and which version of the code was used to extract it (and any related features).



# Testing



- The full ML pipeline is integration tested
- All input feature code is tested
- Model specification code is unit tested
- Model quality is validated before attempting to serve it



# Infrastructure



- Models are tested via a shadow and/or canary process before they enter production serving environments
- Monitor the model performance



# Run Through the Checklist

It's also useful to observe the links and dependencies across different best practices:

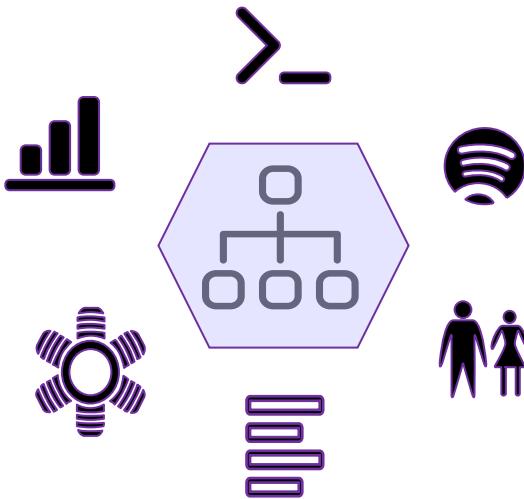
- Without model specification review and version control, it would be hard for reproducible training
- Without reproducible training, the effectiveness and predictability of canary releases are significantly reduced.
- Without knowing the impact of model staleness, it's hard to implement effective monitoring





# Architecture Approaches for ML Systems

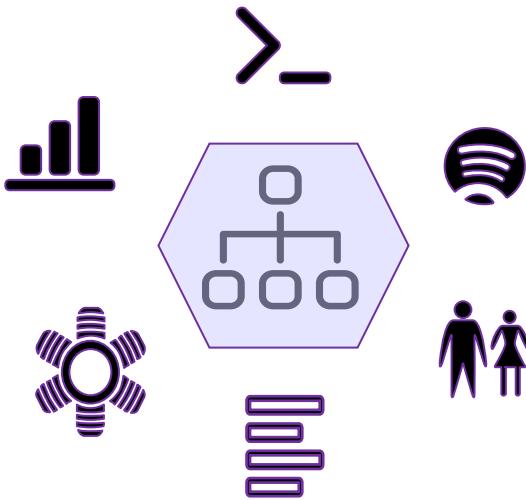
# ML System Architectures



1. Model embedded in application
2. Served via a dedicated service
3. Model published as data (streaming)
4. Batch prediction (offline process)



# Serving ML Models - Formats



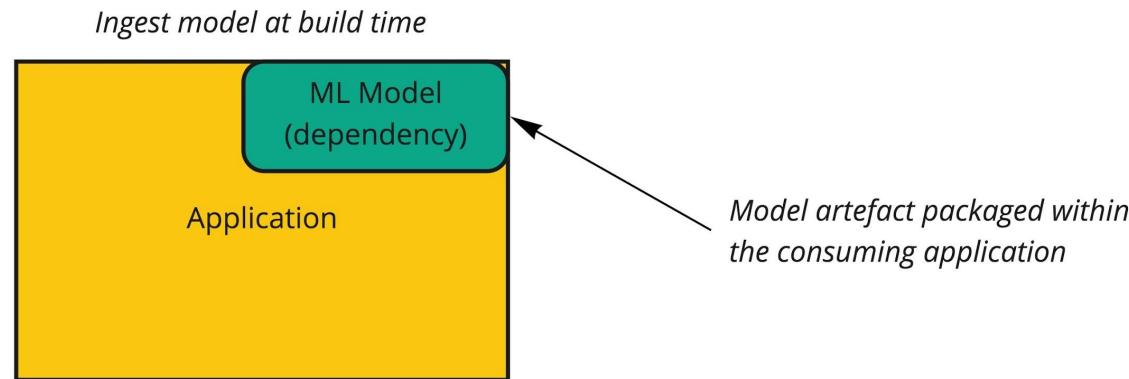
- Serializing the model object with pickle.
- MLFlow (MLeap module) provides a common serialization format for exporting/importing Spark, Scikit-learn, and Tensorflow models.
- Language-agnostic exchange formats to share models, such as PMML, PFA, and ONNX.

# Architecture 1: Embedded

**Pre-Trained:** Yes

**Predict-on-the-fly:** Yes

**Variations:** Embedded on mobile device (e.g. Core ML), running in the browser (Tensorflow.js)

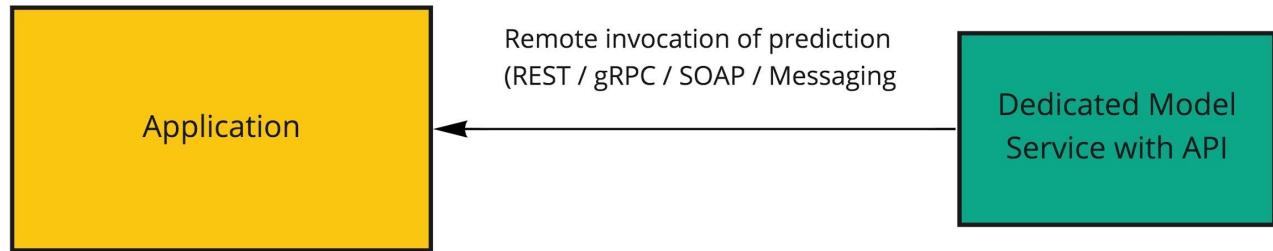


# Architecture 2: Dedicated Model API

Pre-Trained: Yes

Predict-on-the-fly: Yes

Variations: Many. See  
also Architecture 3



*Model is wrapped in a service that  
can be deployed independently*

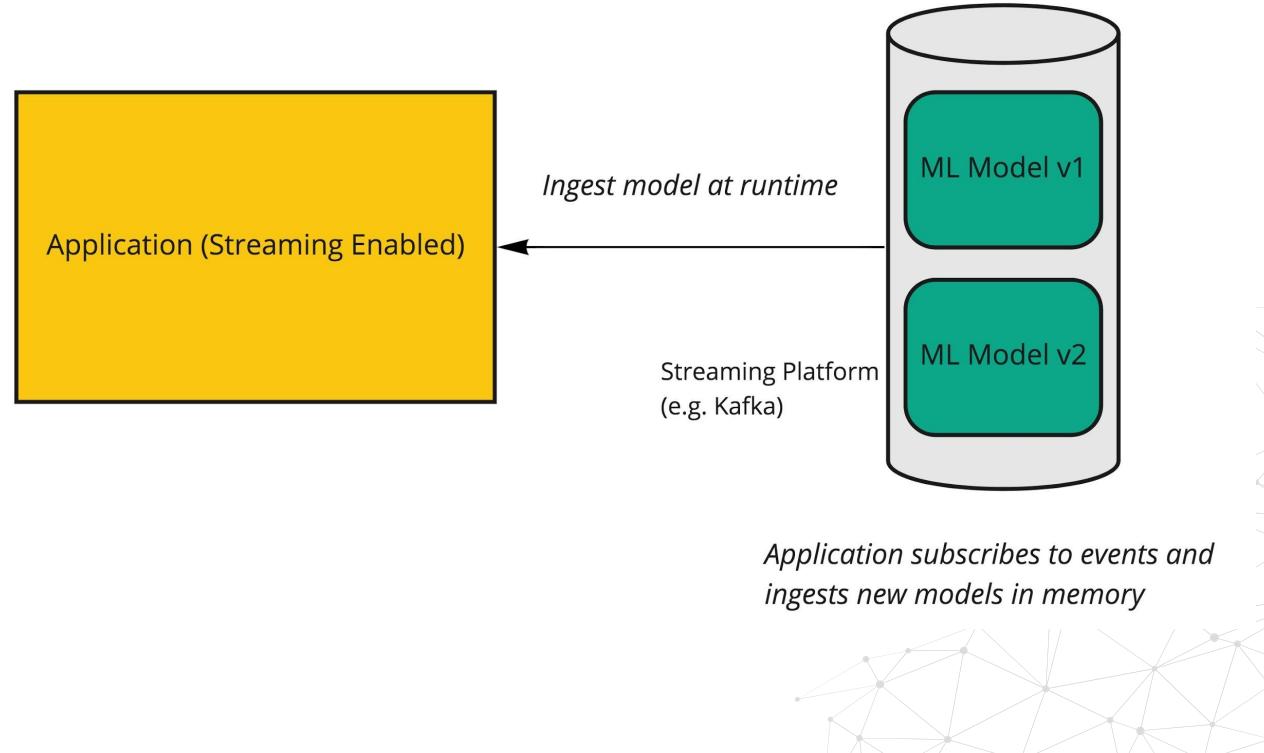


# Architecture 3: Model Published as Data

Pre-Trained: Yes

Predict-on-the-fly: Yes

Variations: Different  
publish/subscribe  
patterns

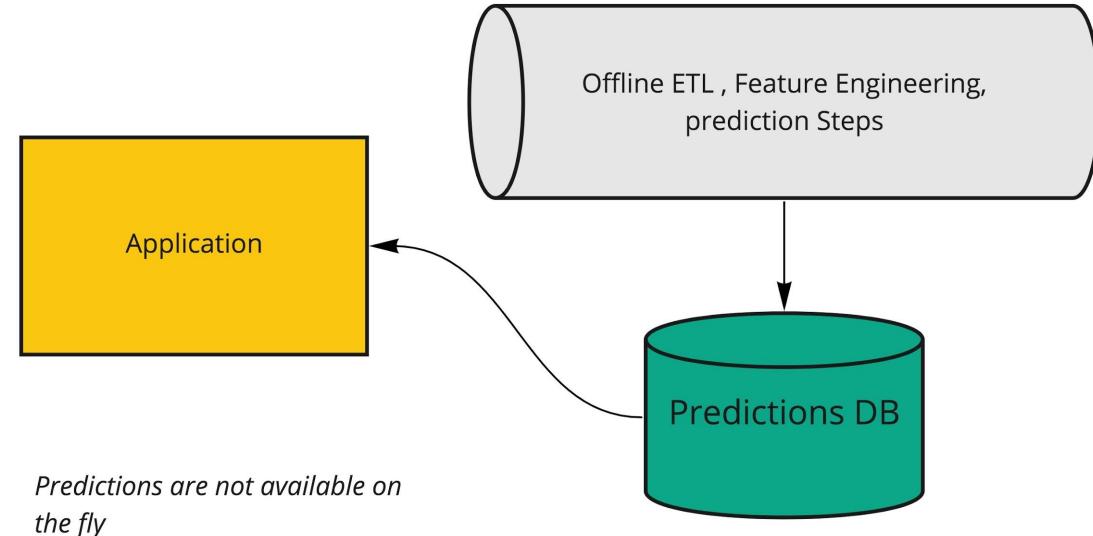


# Architecture 4: Offline Predictions

**Pre-Trained:** Yes

**Predict-on-the-fly:** No

**Variations:** Serve predictions via API, CSV, dashboards



# Architecture Comparison

	Pattern 1 (Embedded)	Pattern 2 (API)	Pattern 3 (Streaming)	Pattern 4 (Offline)
Prediction	On the fly	On the fly	On the fly	Batch Offline
Prediction result delivery	Within app process	Via API	Streaming via Message Queue	Shared DB, API, file
Latency for prediction	Low	Moderate	Depends	Hours/Days
System Management Difficulty	So so	Easy	Very Hard	So so
Model Update requires deployment?	Yes	Yes (of model service)	No	Yes

(1) and (2) are the focus of this course





# Architecture Component Breakdown

(ML model as embedded dependency,  
predict on the fly)

# High Level Architecture

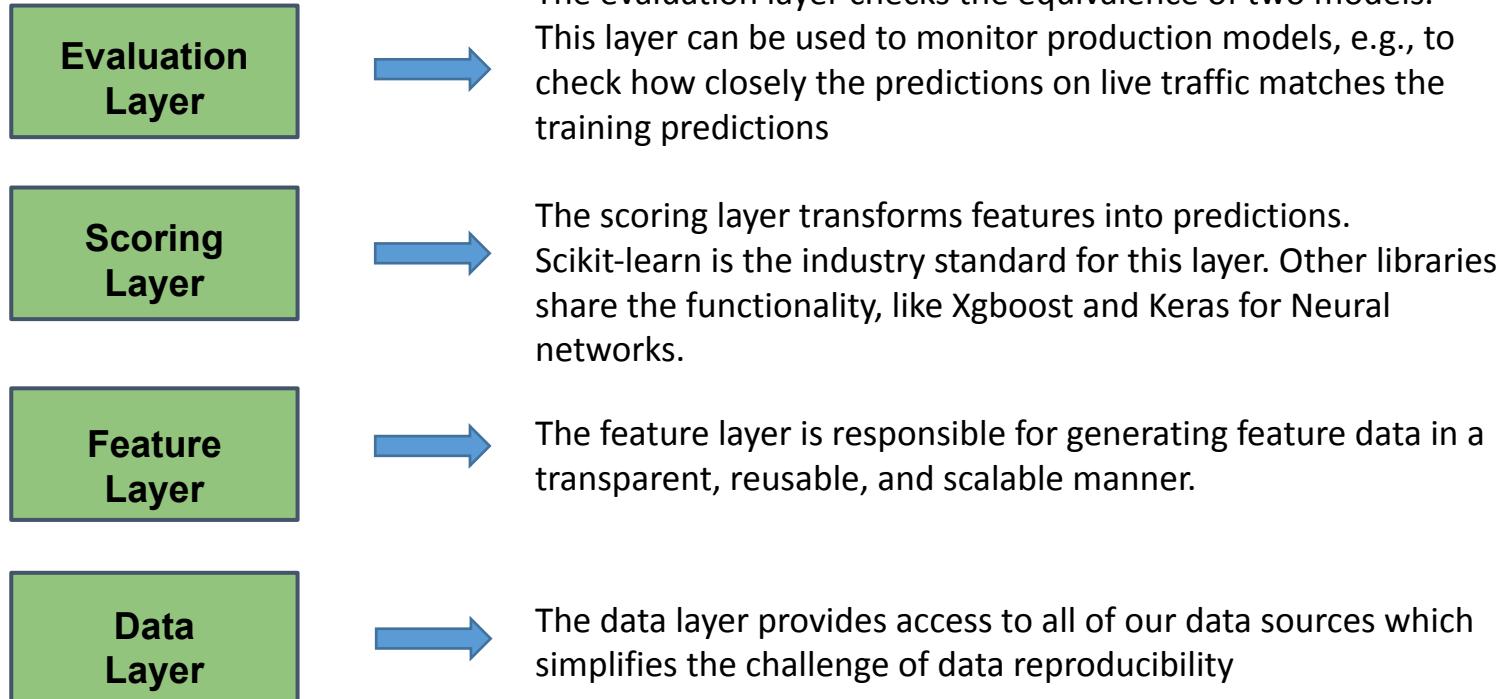
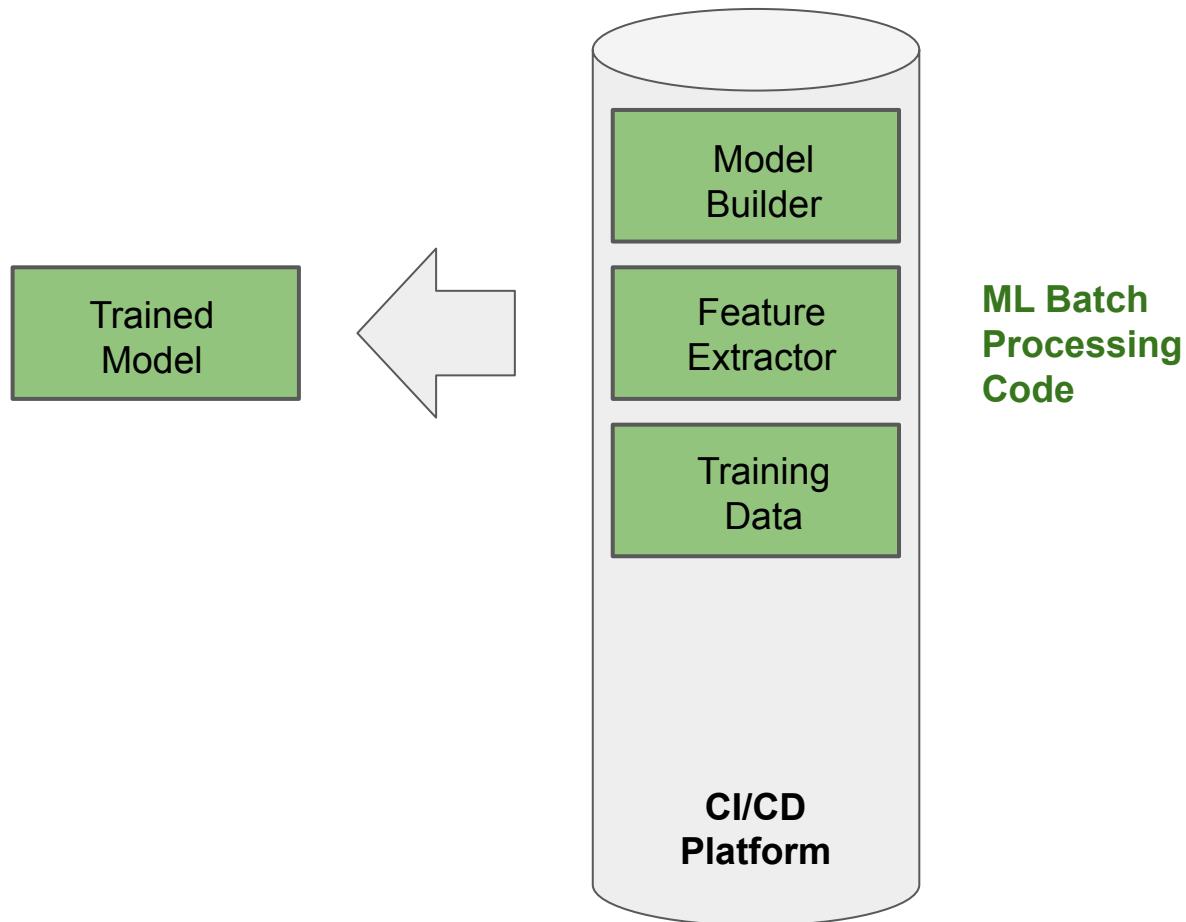
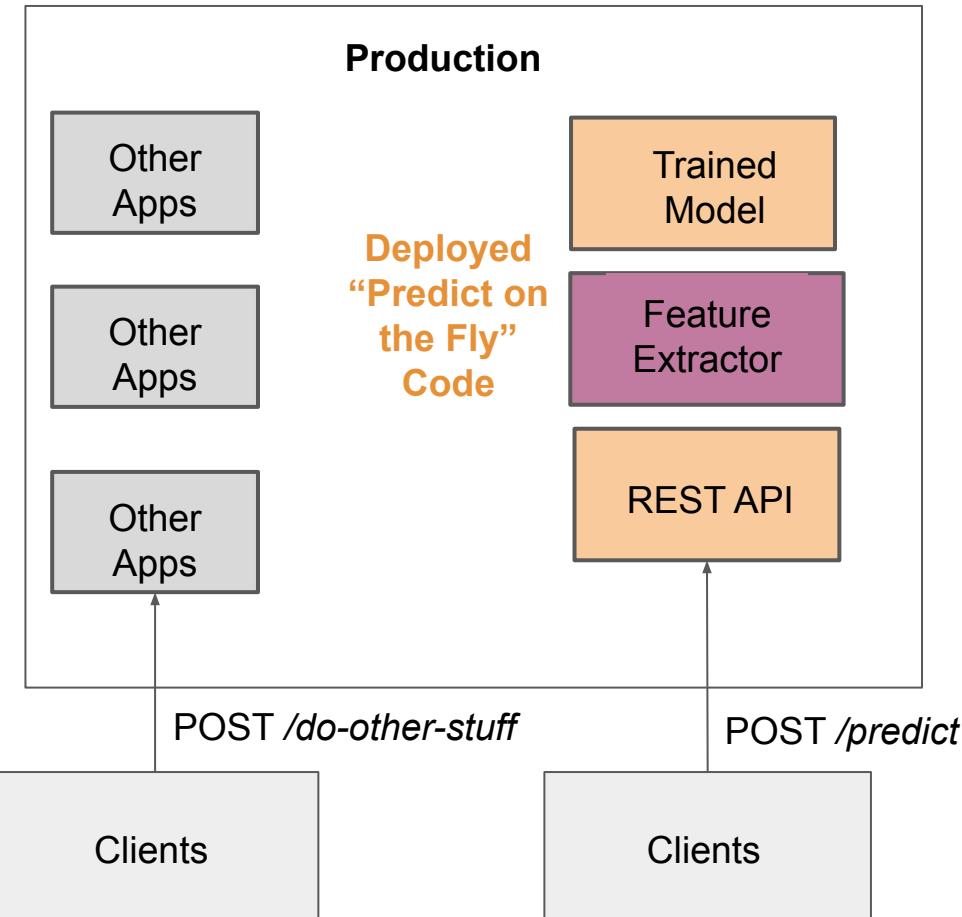


Diagram taken from:  
[Building a Reproducible Machine Learning Pipeline](#), Sugimura and Hartl

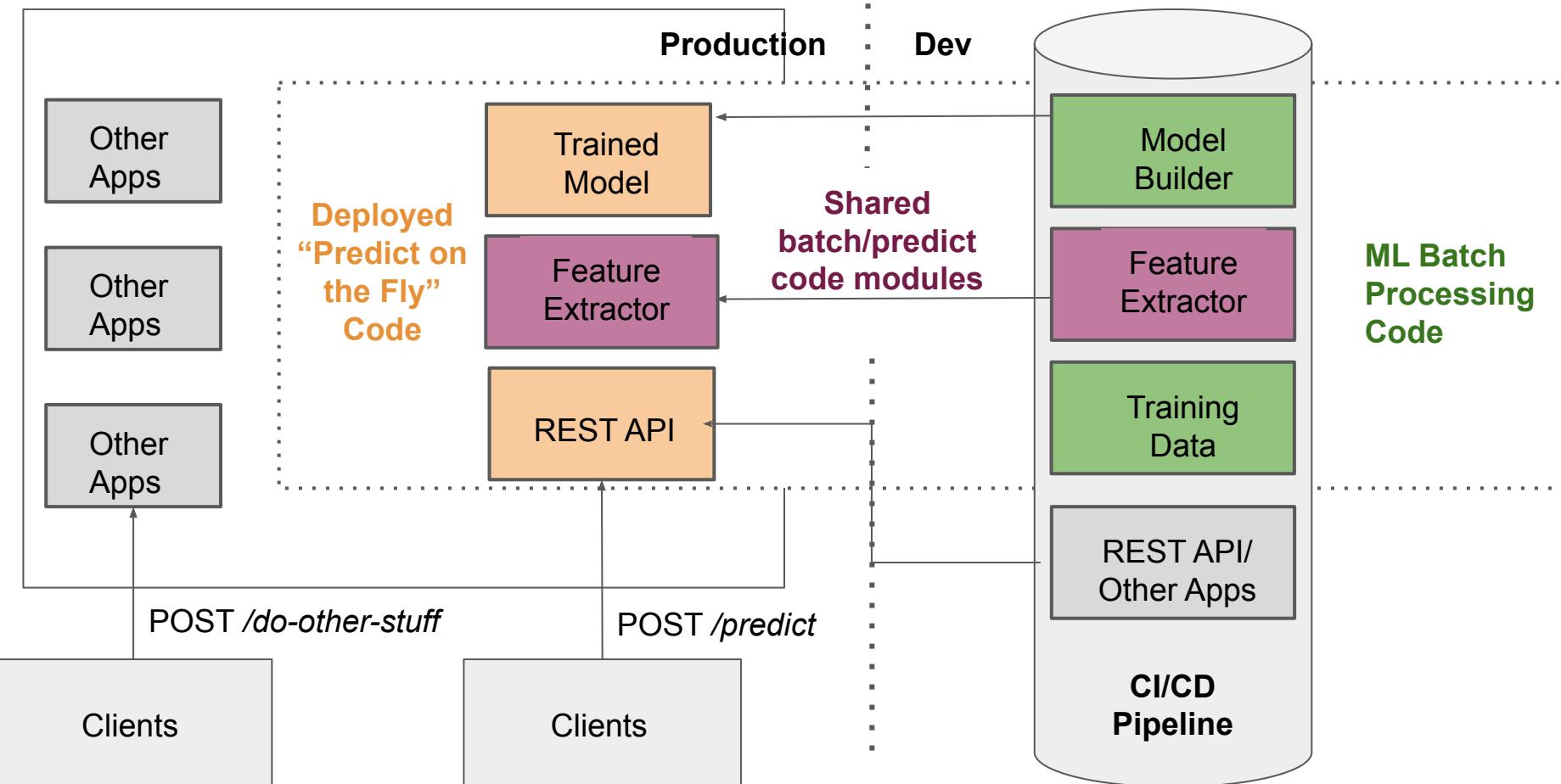
# Breakdown: Training Phase



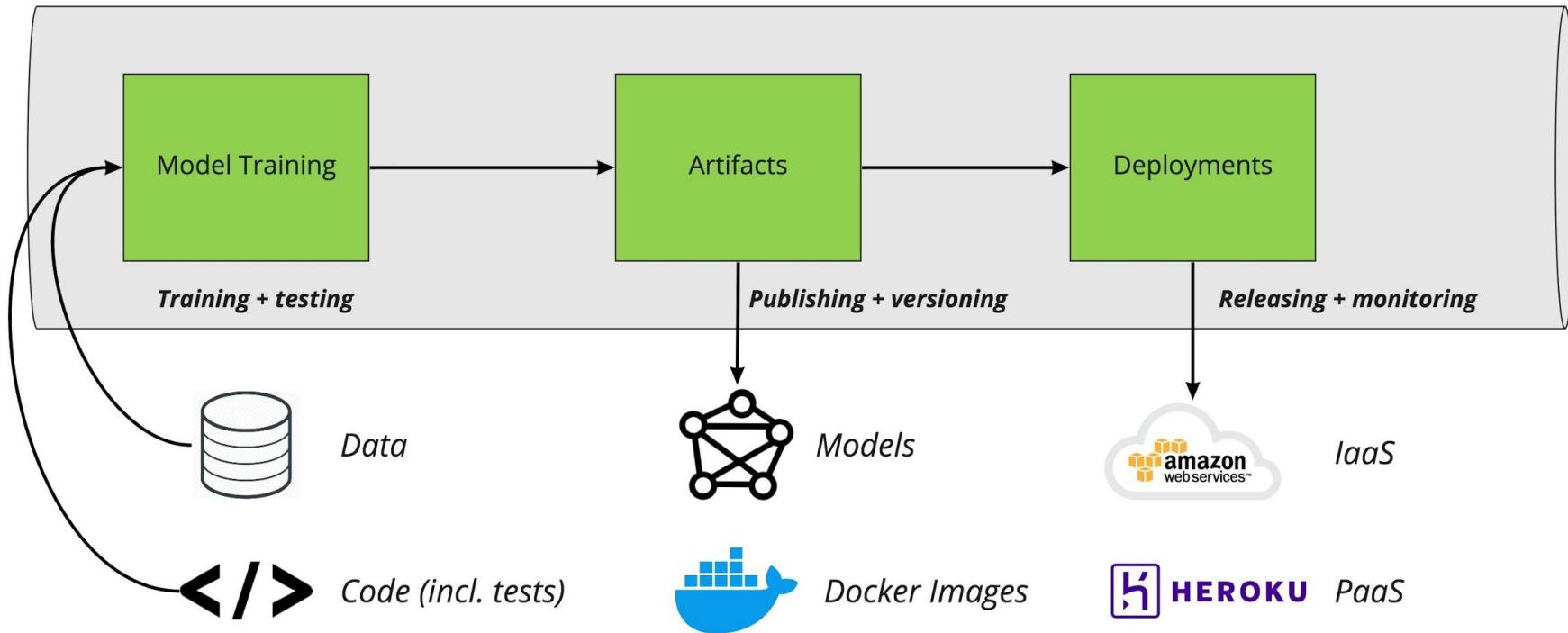
# Breakdown: Prediction Phase



# Diagram: Train by batch, predict on the fly, serve via REST API



## ***CI/CD Automation of the Deployment***



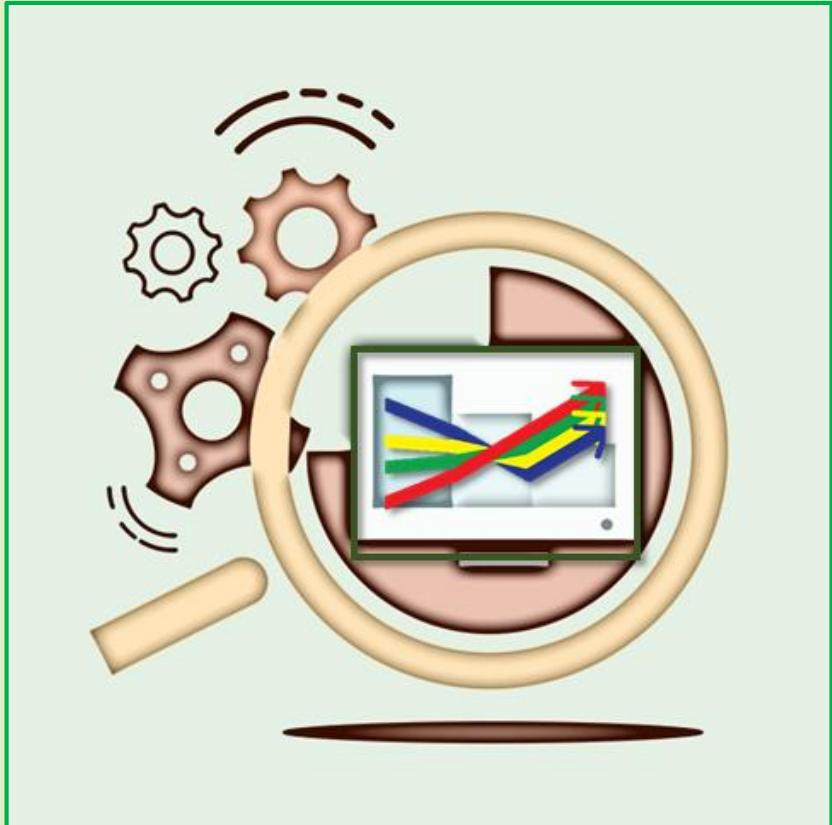


# THANK YOU



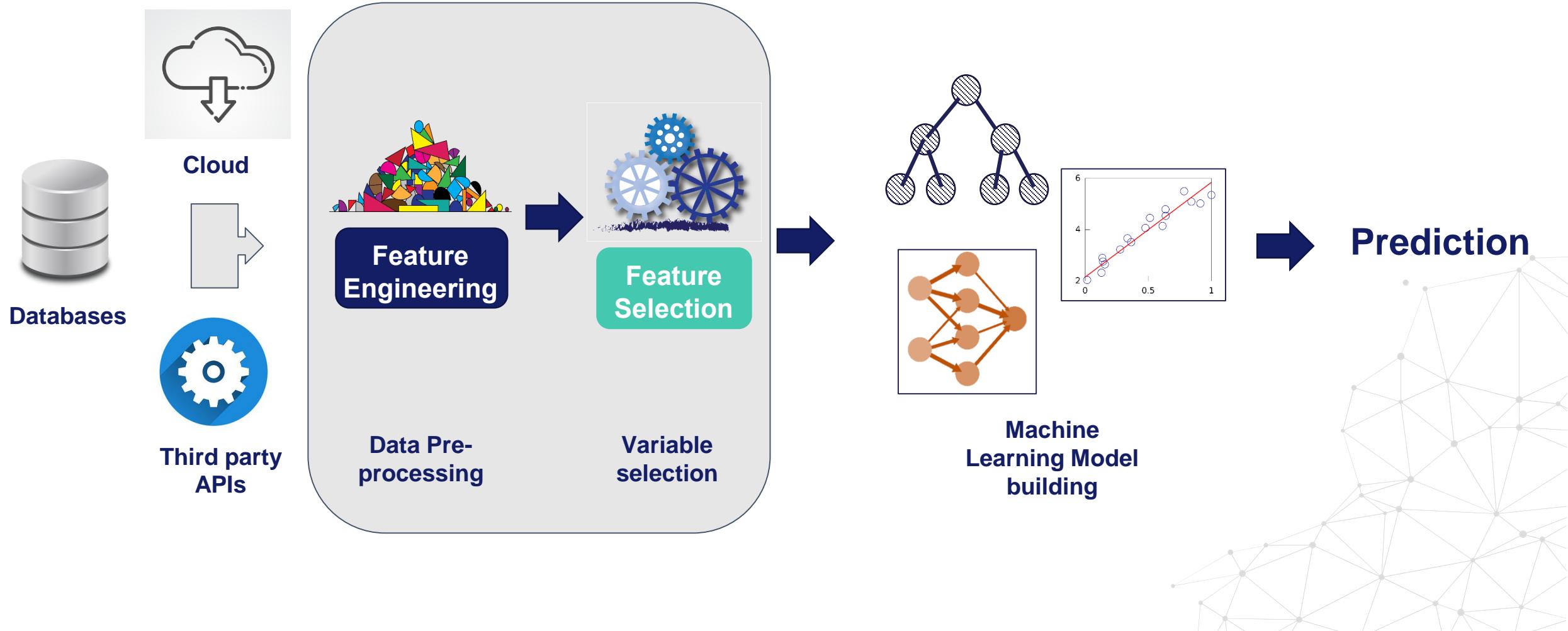
I tweet about software engineering + build in public  
@ChrisSamiullah



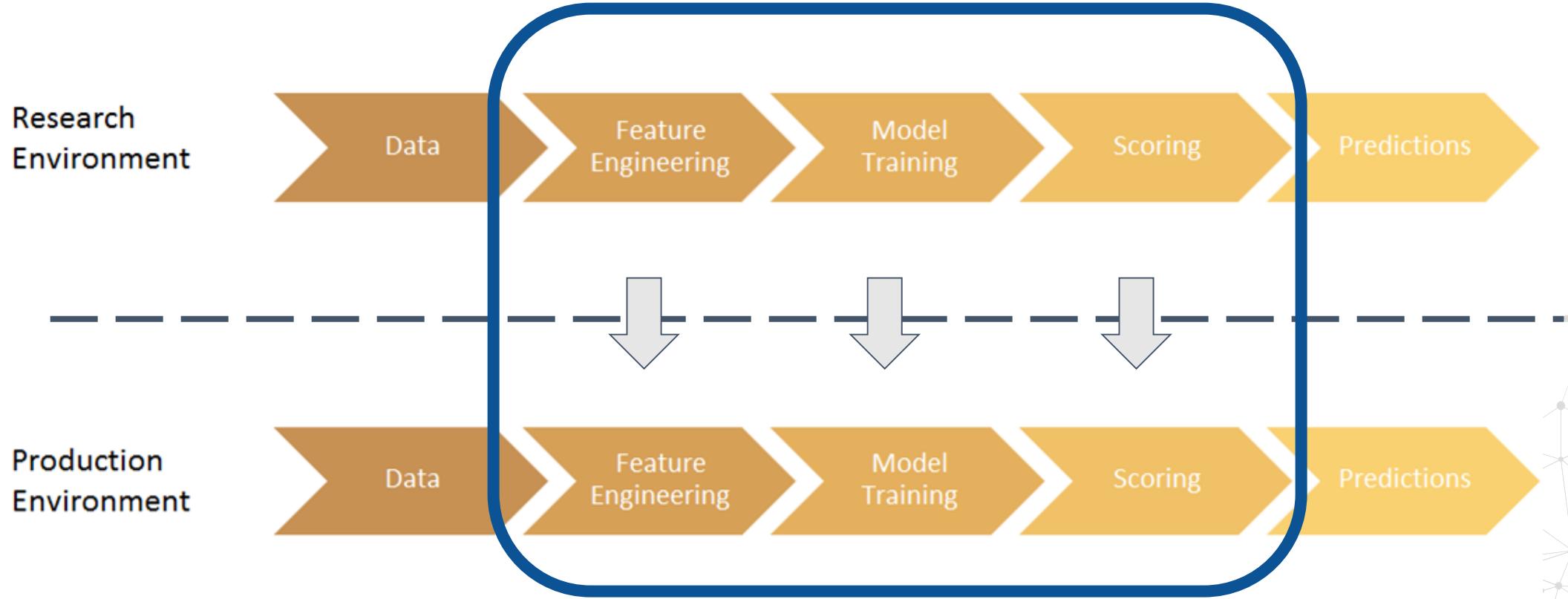


# Research Environment - Creating a ML Model

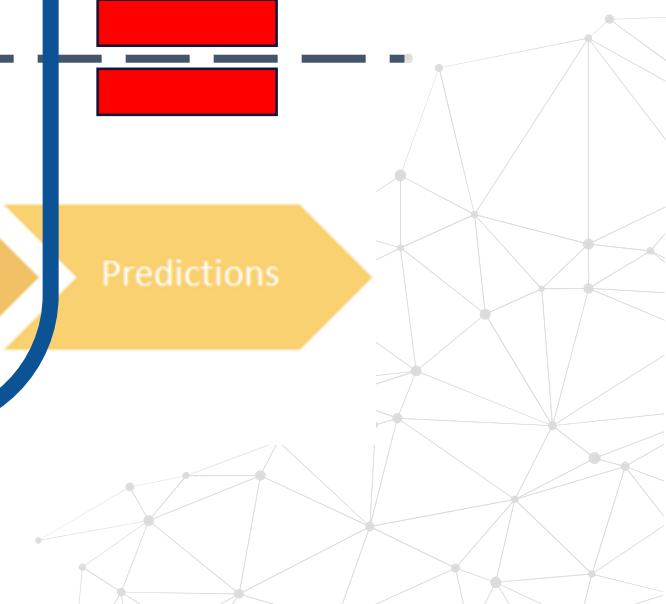
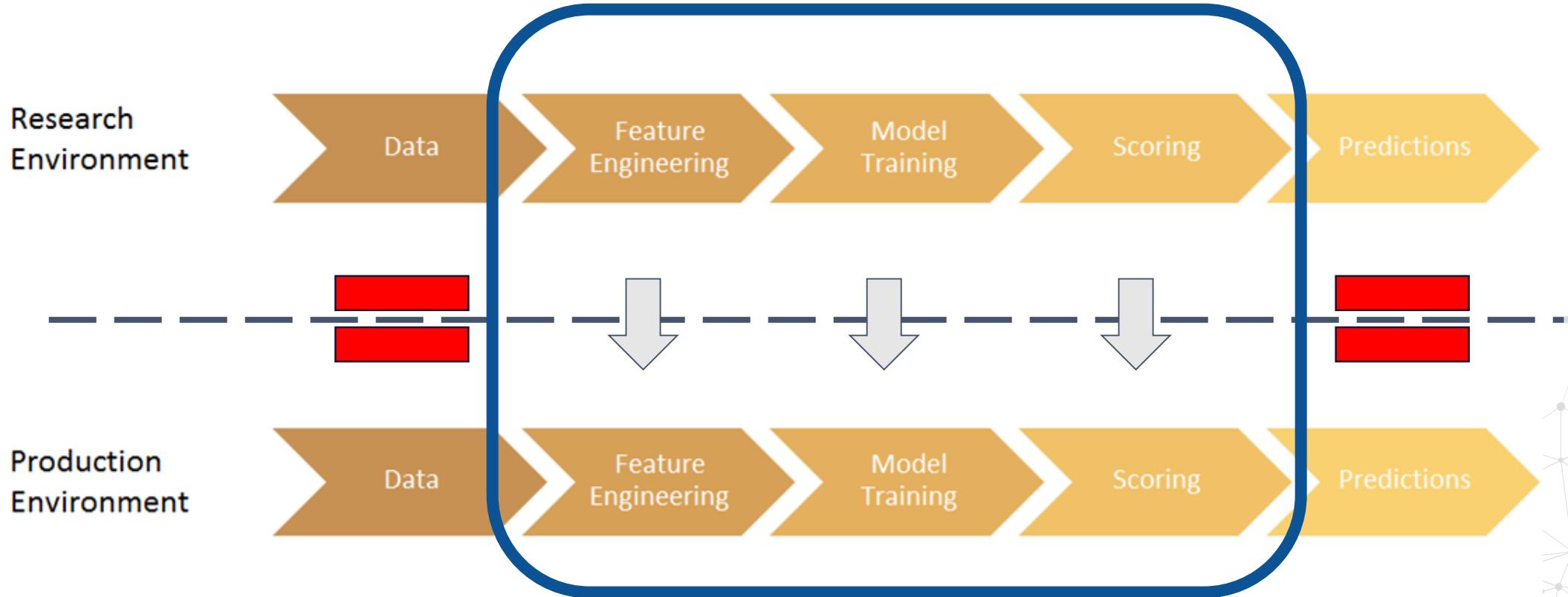
# Machine Learning Pipeline



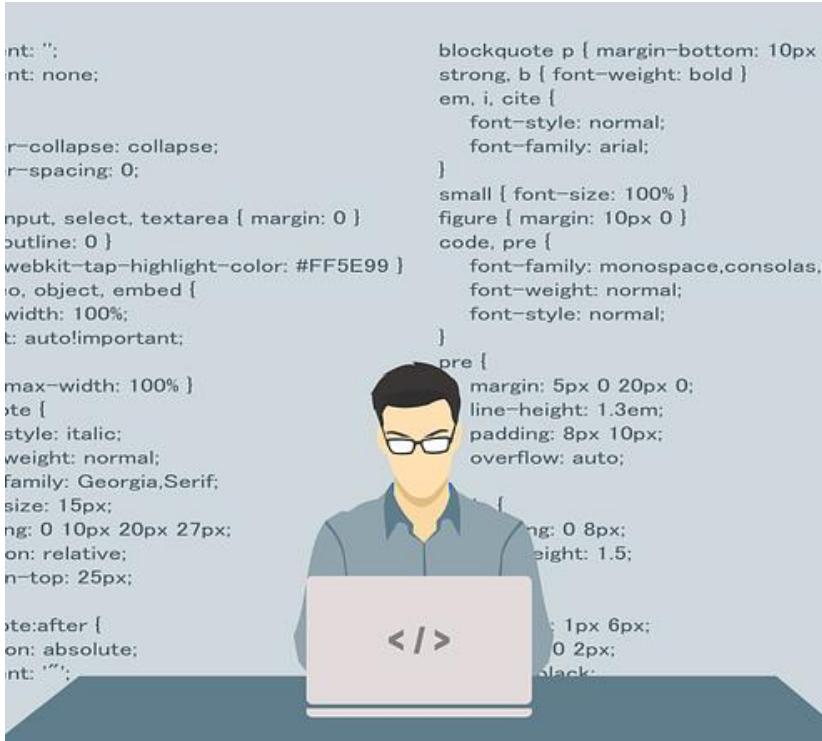
# Deployment of ML Pipeline



# Reproducibility



# Principles of ML Deployment



- Challenges of traditional software

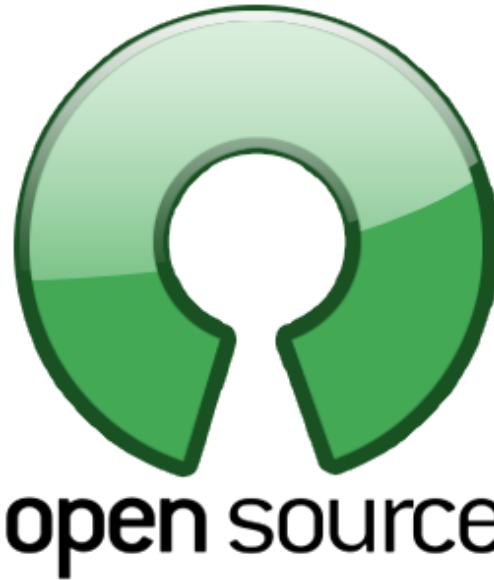
- Reliability
- Reusability
- Maintainability
- Flexibility

- Additional challenges specific to Machine Learning

- Reproducibility



# • Open-source



- Increase Performance
  - ✓ Ready to use code
  - ✓ Off-the-shelf algorithms
- Maximise reproducibility
  - ✓ Versioning
- Reliability
  - ✓ Testing
- Reusability, maintainability
- Minimise deployment times



# In-house software



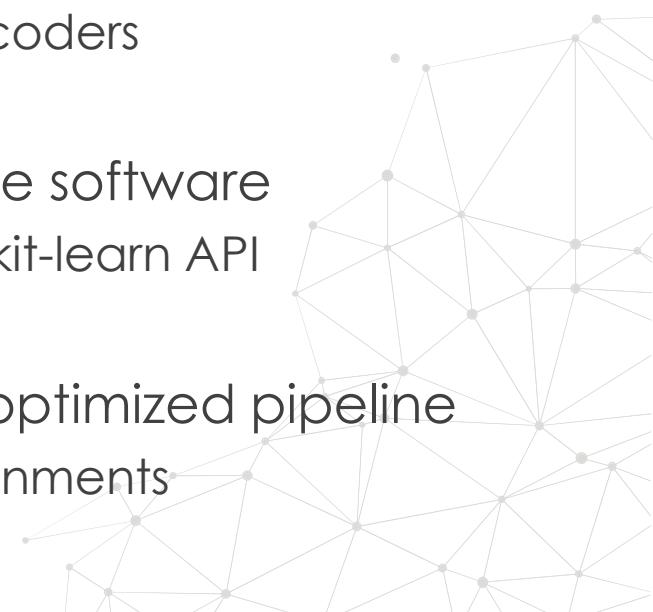
- Versioned
  - ✓ Reproducibility
- Tested
  - ✓ Reliability
- Shareable
  - ✓ Reusability
- Minimise deployment times



# • 5 STEPS



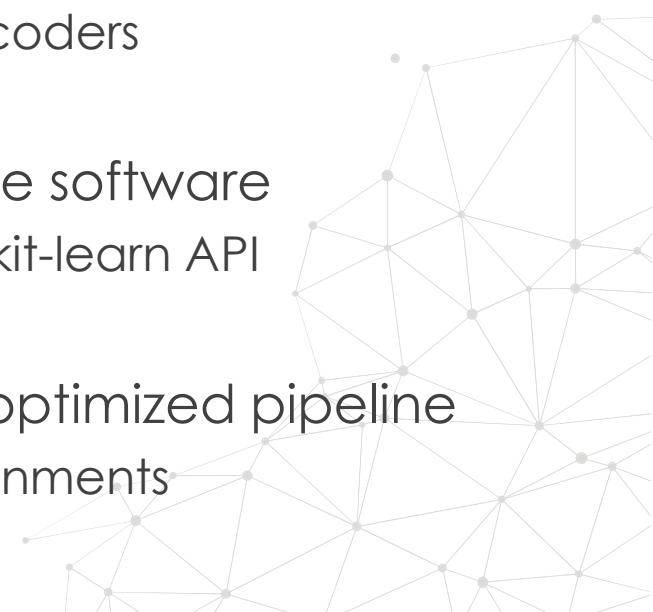
1. Overview of ML steps
  - Feature engineering and selection
  - Model building
2. Create a ML Pipeline
3. Available Open-Source
  - Scikit-learn, Feature-engine, Category encoders
4. Create in house software
  - Use of the scikit-learn API
5. Re-create an optimized pipeline
  - Python environments



# • 5 STEPS



1. **Overview of ML steps**
  - Feature engineering and selection
  - Model building
2. Create a ML Pipeline
3. Available Open-Source
  - Scikit-learn, Feature-engine, Category encoders
4. Create in house software
  - Use of the scikit-learn API
5. Re-create an optimized pipeline
  - Python environments





# THANK YOU

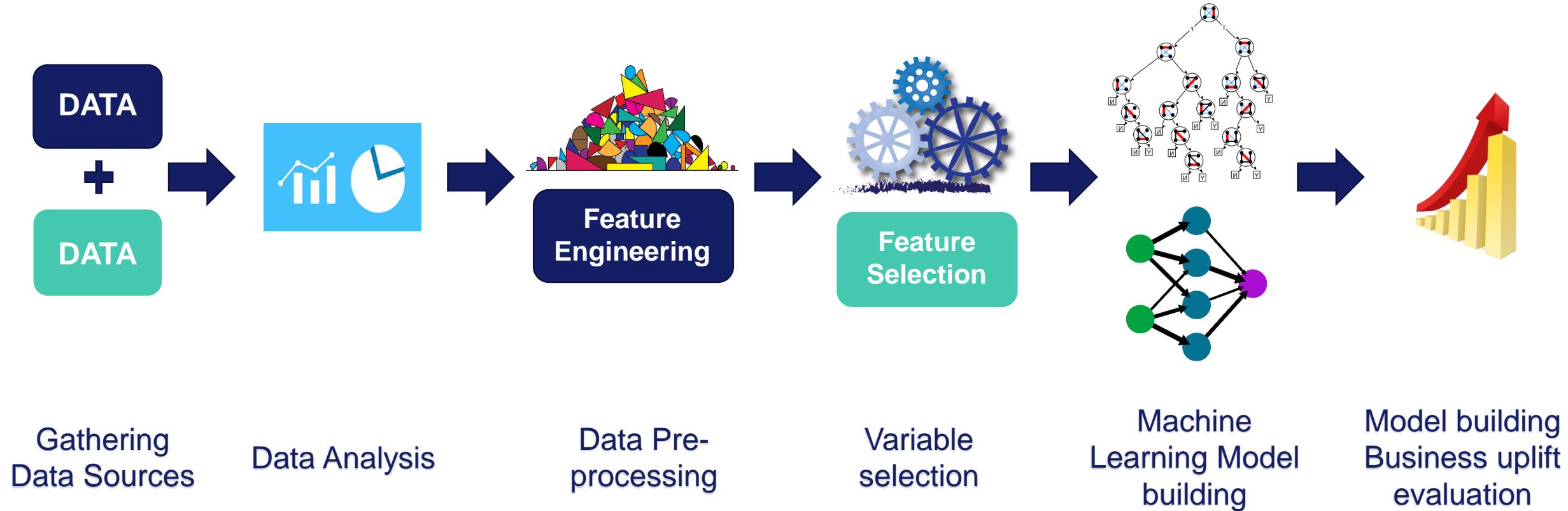
[www.trainindata.com](http://www.trainindata.com)



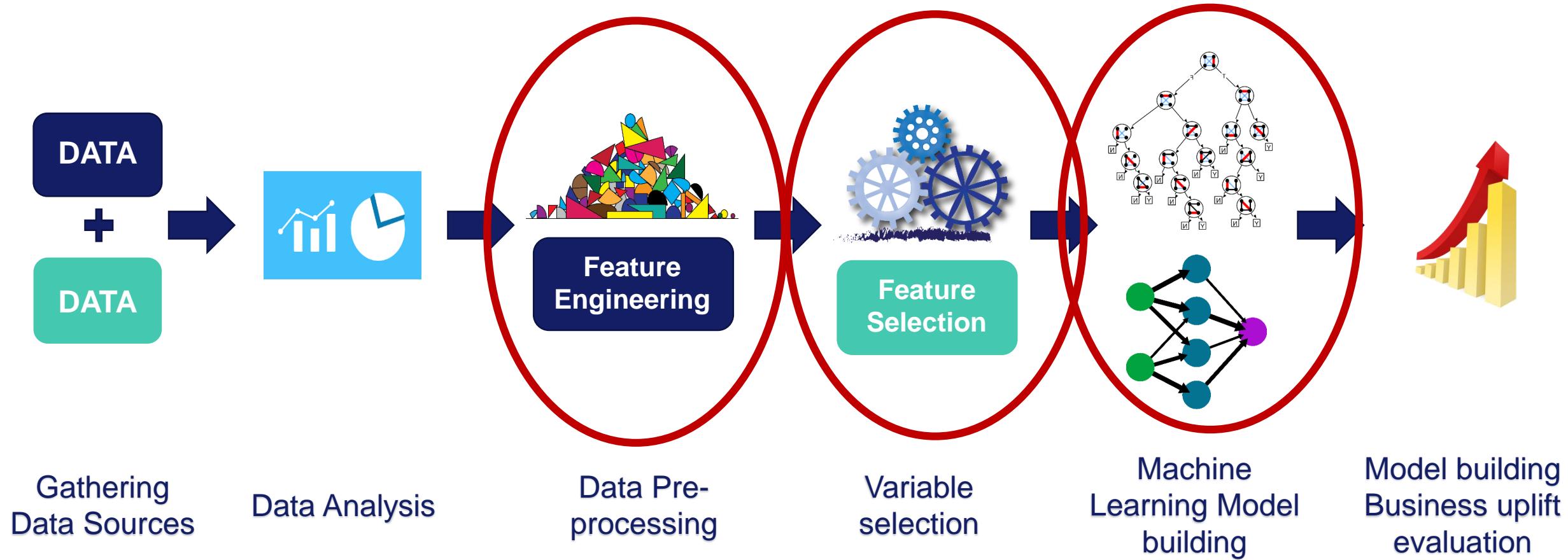
# Machine Learning Model Pipeline Overview



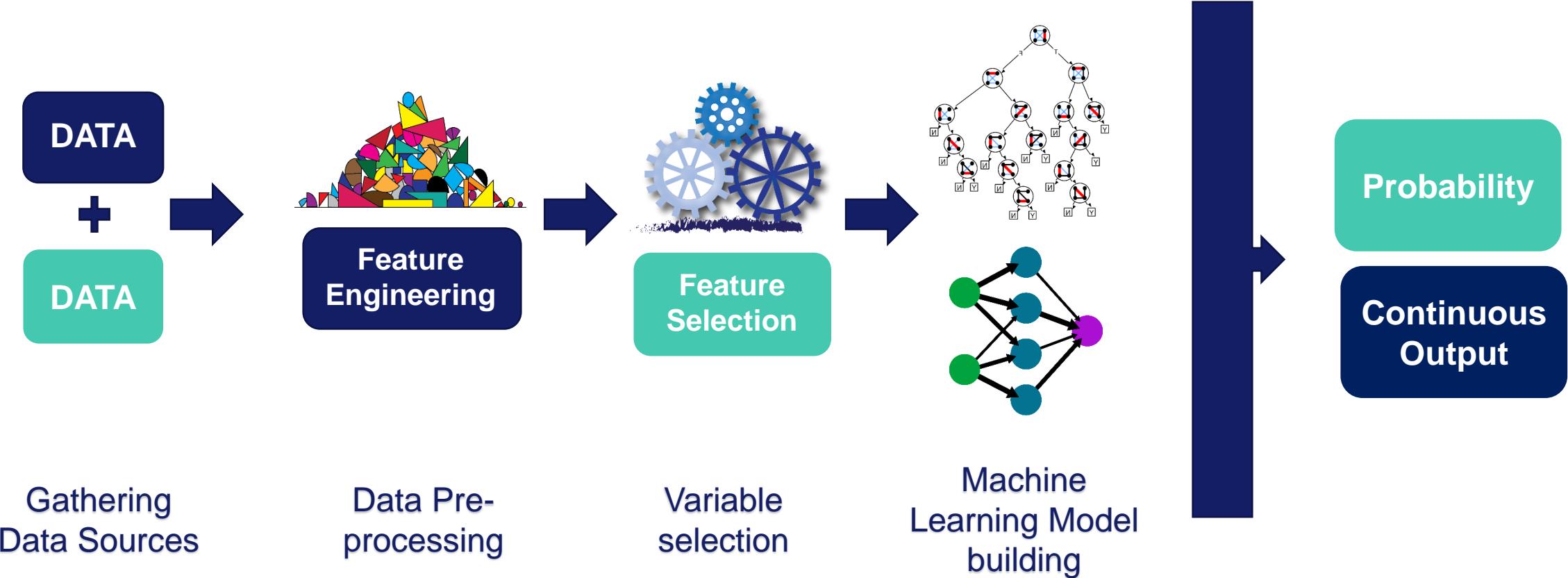
# Machine Learning Pipeline: Overview



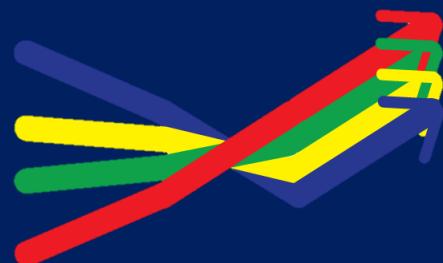
# Machine Learning Pipeline: Production



# Machine Learning Pipeline: Production



# Feature Engineering – Variable Transformations

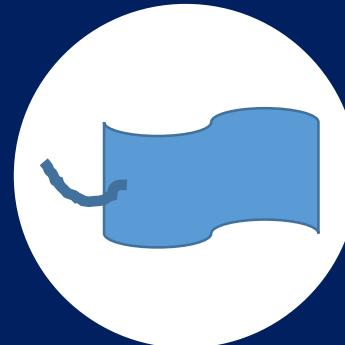


# Feature Engineering



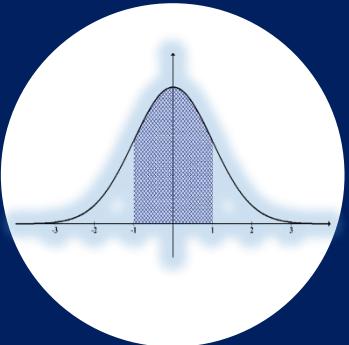
## Missing data

Missing values within  
a variable



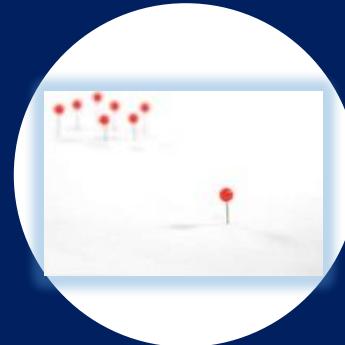
## Labels

Strings in  
categorical  
variables



## Distribution

Normal vs skewed

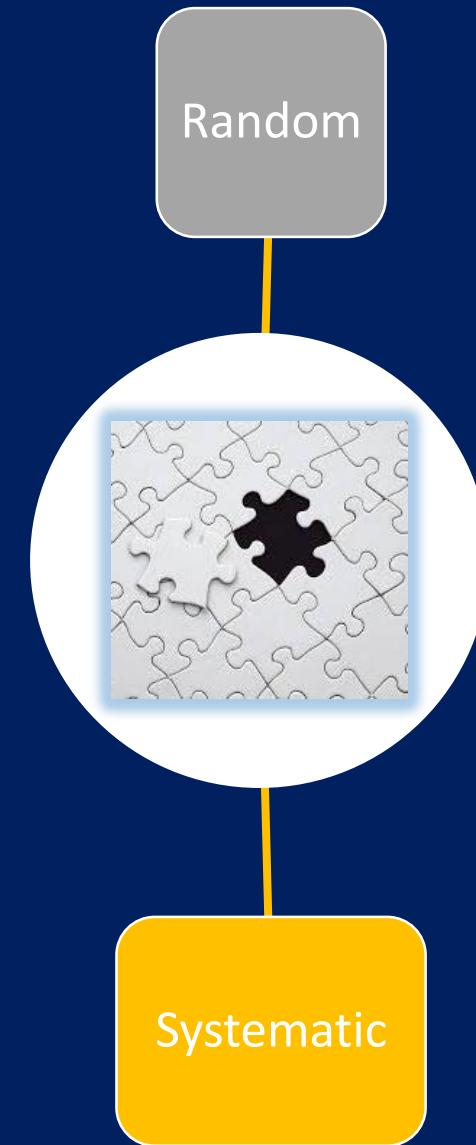


## Outliers

Unusual or  
unexpected values

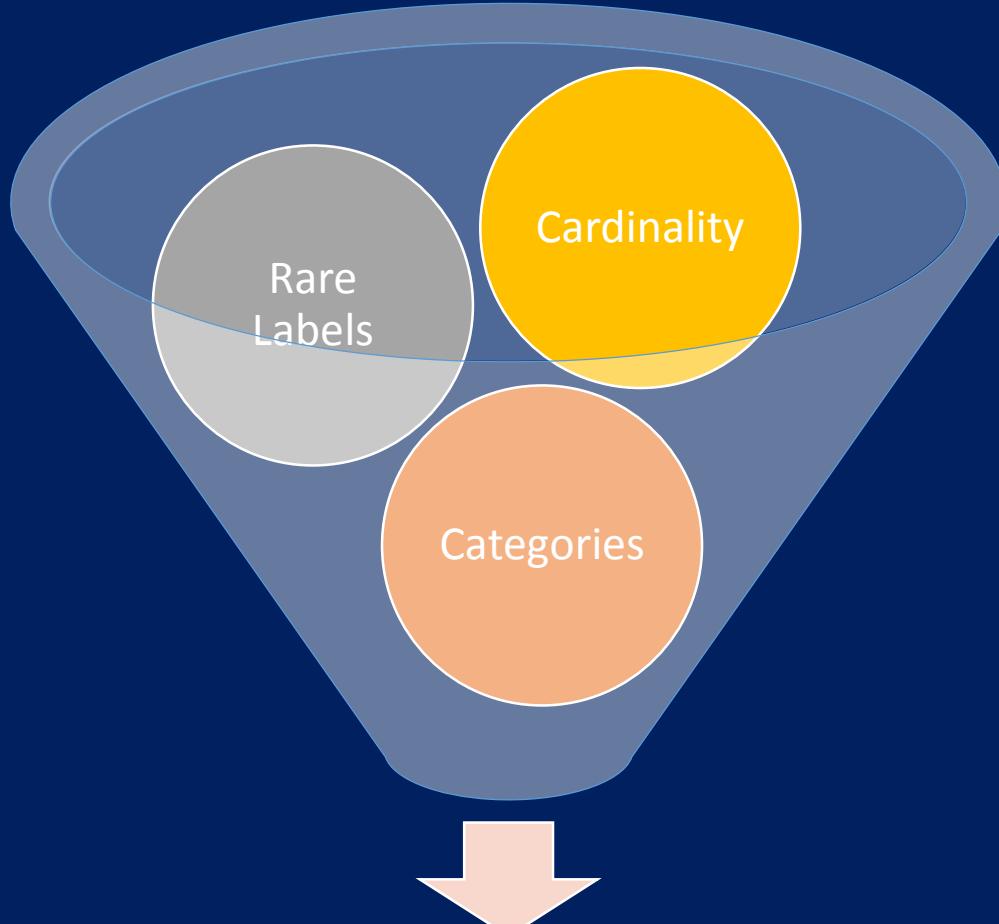
# Missing Data

- Missing values for certain observations
- Affects all machine learning models
  - Scikit-learn

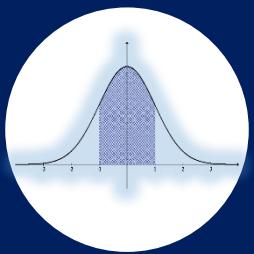


# Labels in categorical variables

- Cardinality: high number of labels
- Rare Labels: infrequent categories
- Categories: strings
  - Scikit-learn



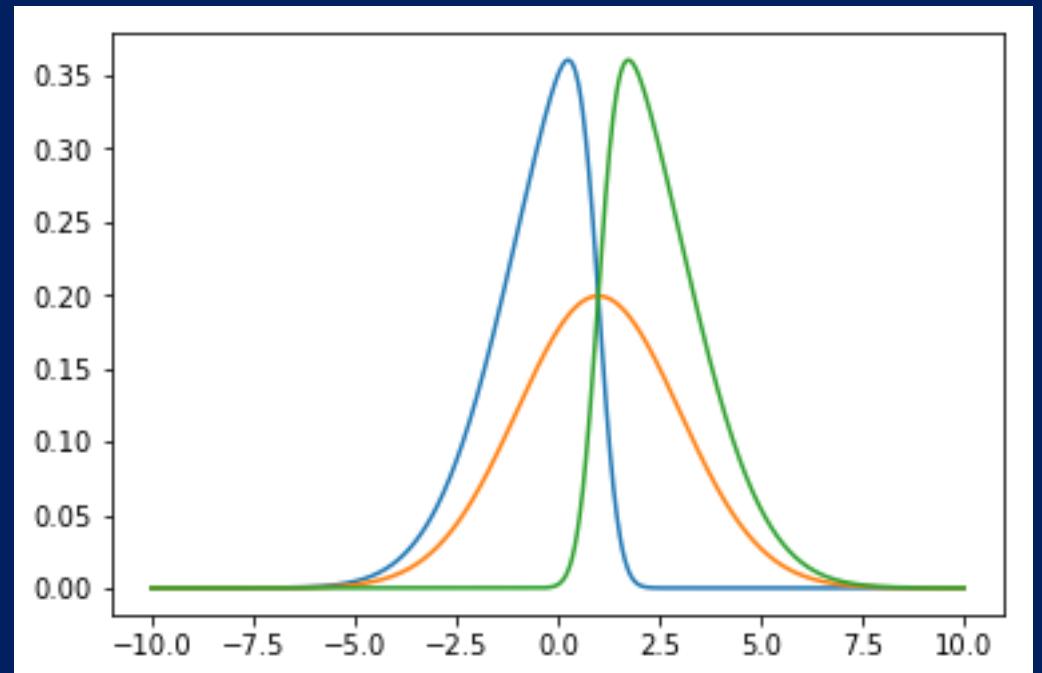
Overfitting in tree based  
algorithms



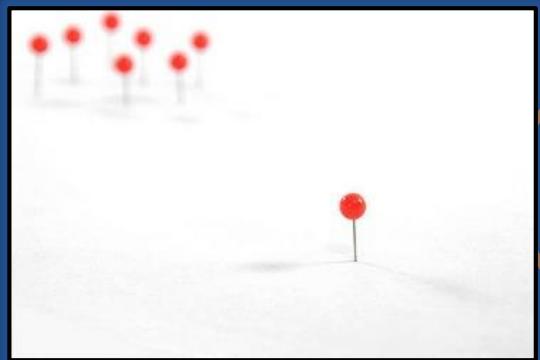
# Distributions

- Better spread of values may benefit performance

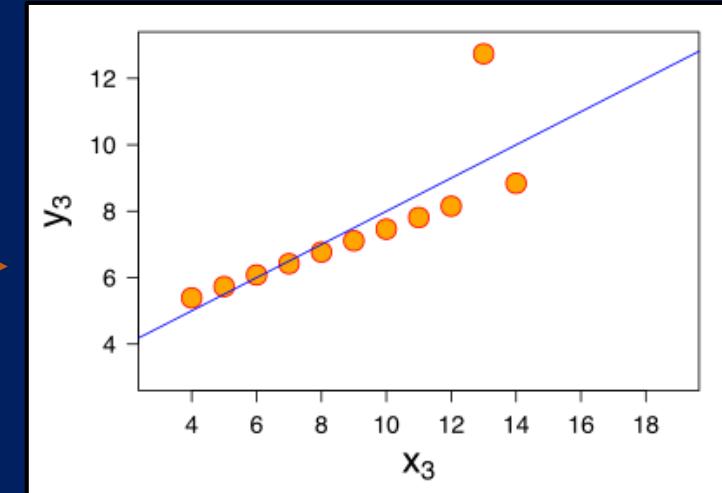
Gaussian vs Skewed



# Outliers



Linear  
models



Adaboost

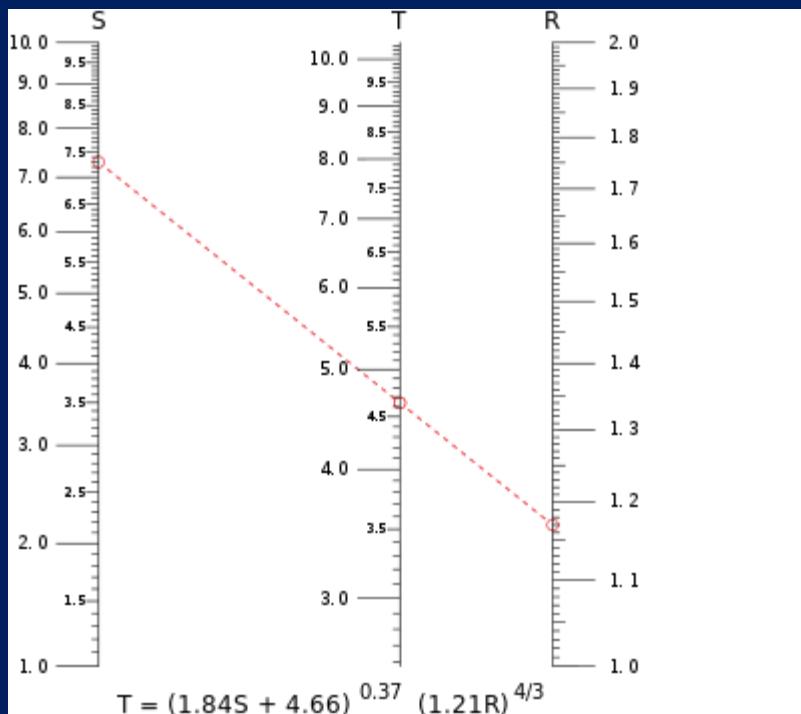


Tremendous  
weights



Bad  
generalisation

# Feature Magnitude - Scale



## Machine learning models sensitive to feature scale:

- Linear and Logistic Regression
- Neural Networks
- Support Vector Machines
- KNN
- K-means clustering
- Linear Discriminant Analysis (LDA)
- Principal Component Analysis (PCA)

## Tree based ML models insensitive to feature scale:

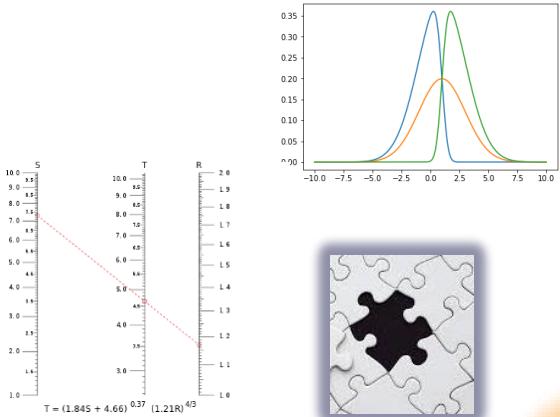
- Classification and Regression Trees
- Random Forests
- Gradient Boosted Trees



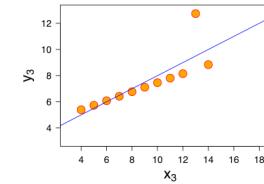
# Feature Engineering



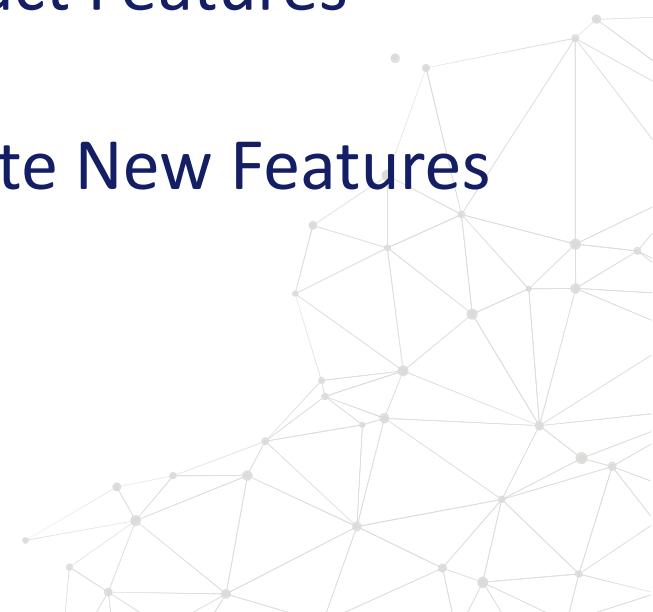
# Feature Engineering



An insurance claim  
A formal request to an insurance company for payment based on the terms of an insurance policy.  
Insurance claims are recorded by the insurer and used to calculate premiums.



- Transform Variables
- Extract Features
- Create New Features



# Missing Data

- Scikit-learn and other libraries can't work with missing data



# Missing Data Imputation Techniques

## Numerical Variables



- Mean / Median Imputation
- Arbitrary value imputation
- End of tail imputation

## Categorical Variables

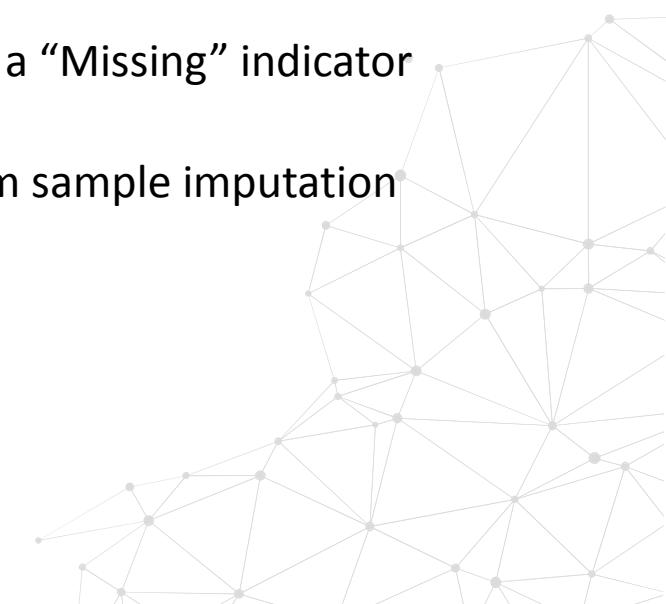


- Frequent category imputation
- Adding a “missing” category

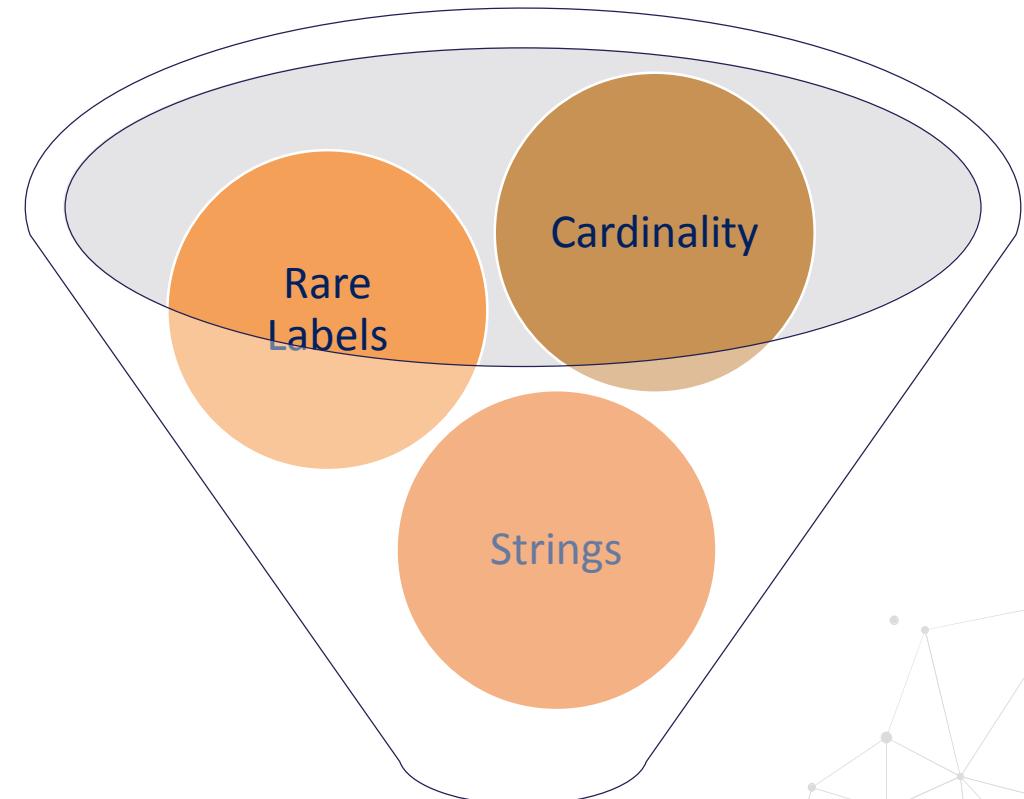
## Both



- Complete Case Analysis
- Adding a “Missing” indicator
- Random sample imputation



# Categorical Variables



Over-fitting

Particularly in decision trees



# Categorical Encoding Techniques

## Traditional techniques

One hot encoding

Count / frequency encoding

Ordinal / Label encoding

## Monotonic relationship

Ordered label encoding

Mean encoding

Weight of evidence

## Alternative techniques

Binary encoding

Feature hashing

Others



# Encoding Techniques: Rare labels



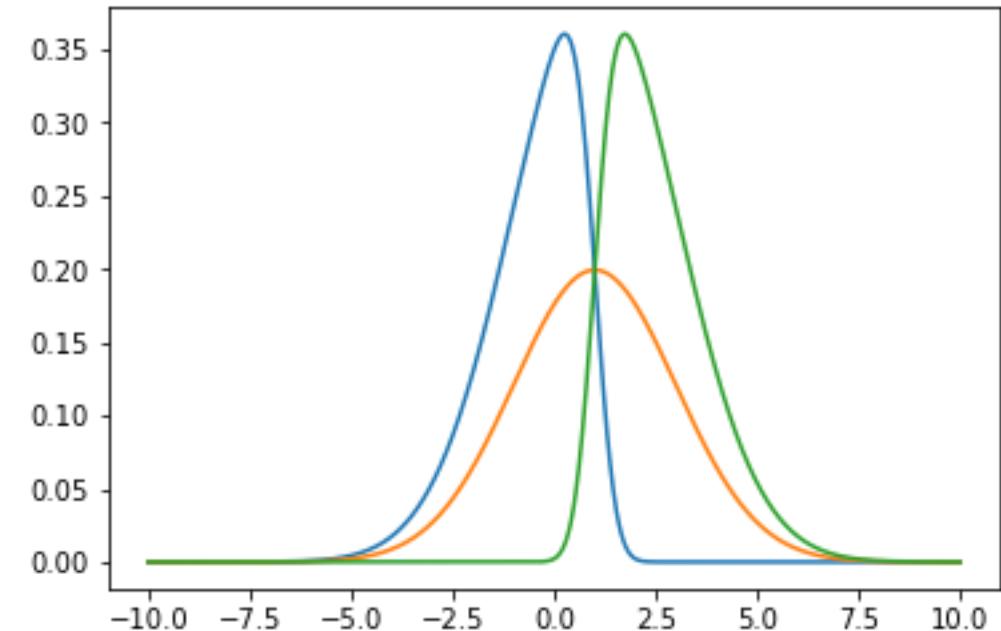
Particularly important for model deployment

- ✓ One hot encoding of frequent categories
- ✓ Grouping of rare categories

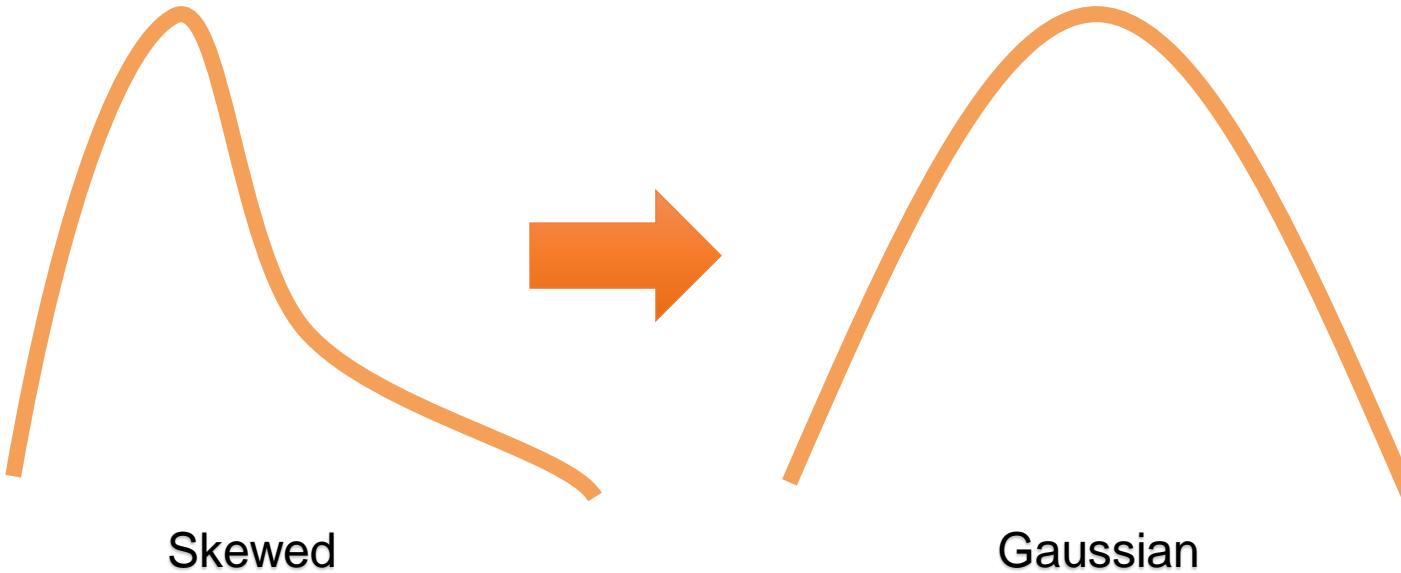


# Distributions

- Some models make assumptions on the variable distributions

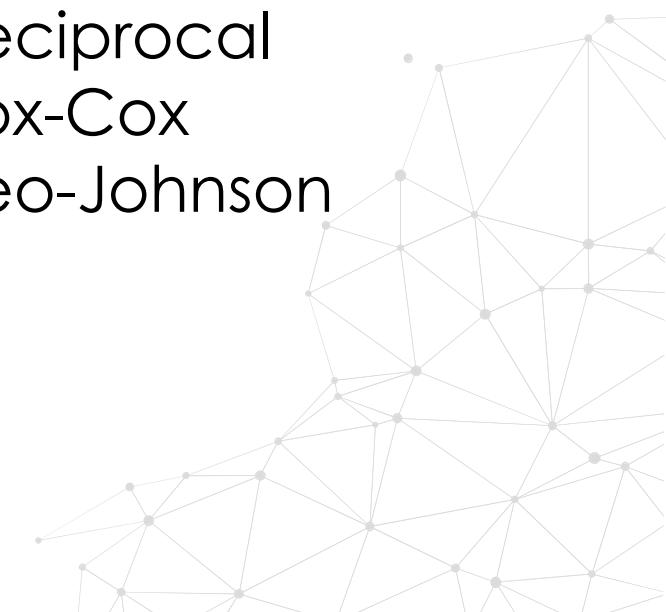


# Mathematical transformations

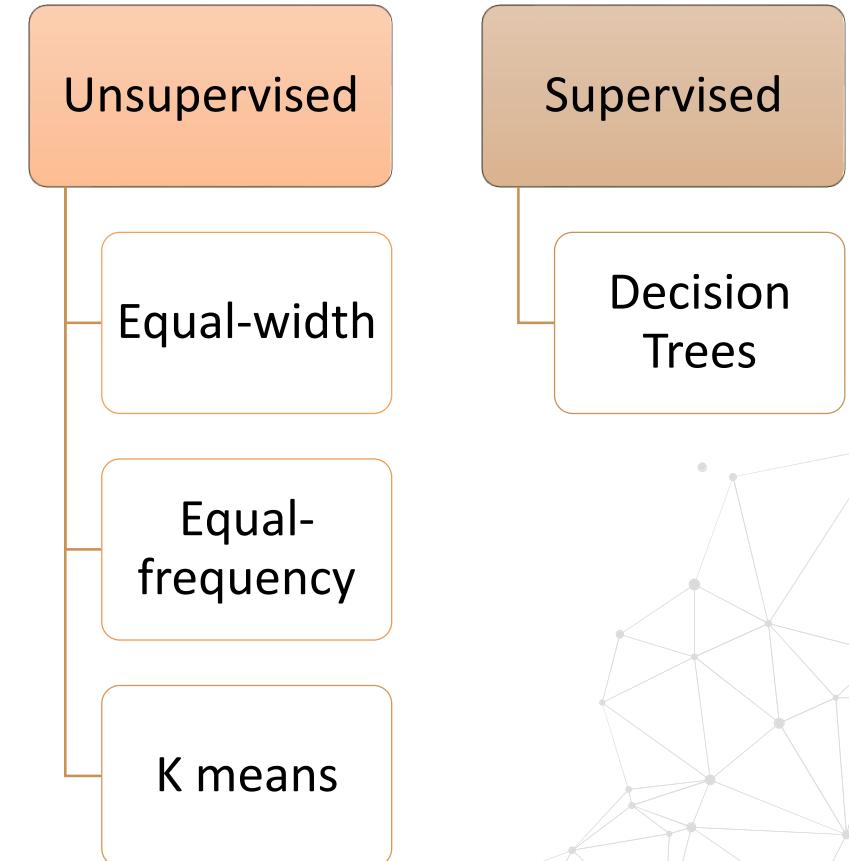
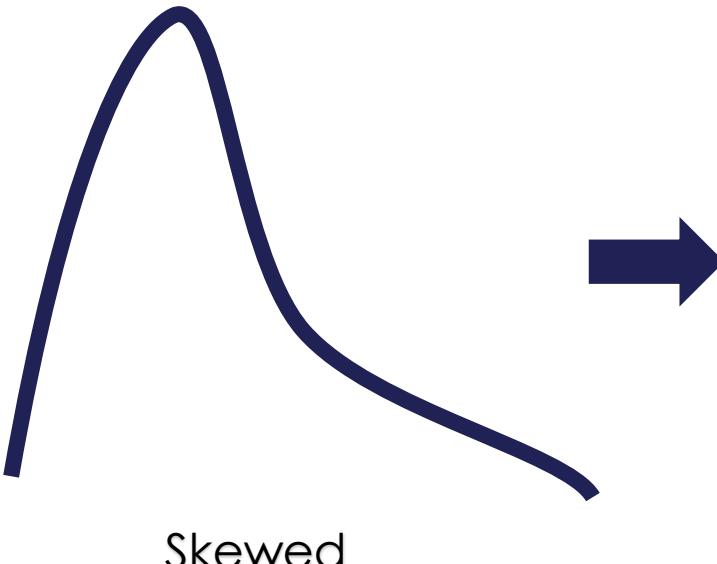


## Variable transformation

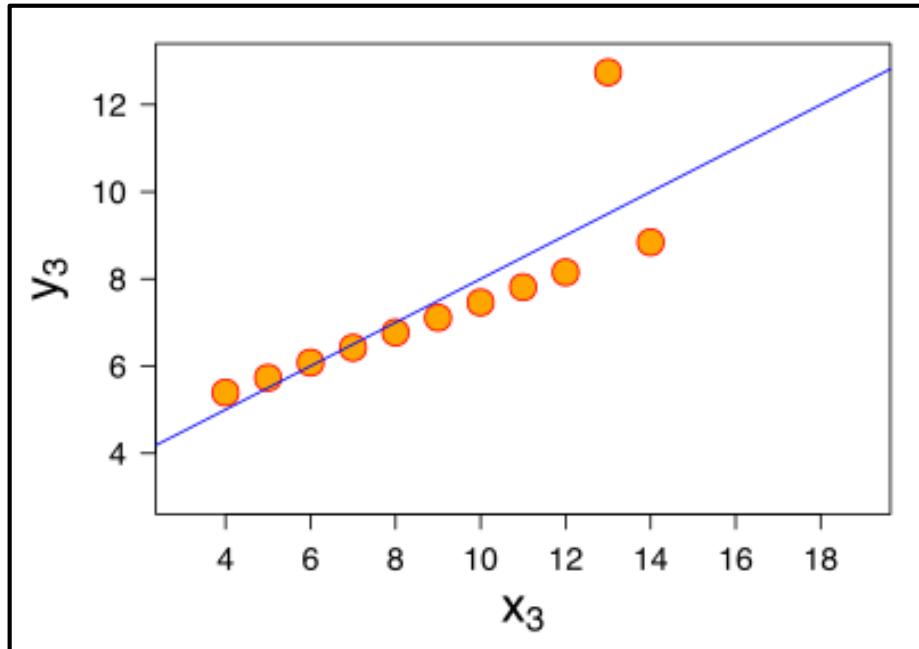
- Logarithmic
- Exponential
- Reciprocal
- Box-Cox
- Yeo-Johnson



# • Discretisation



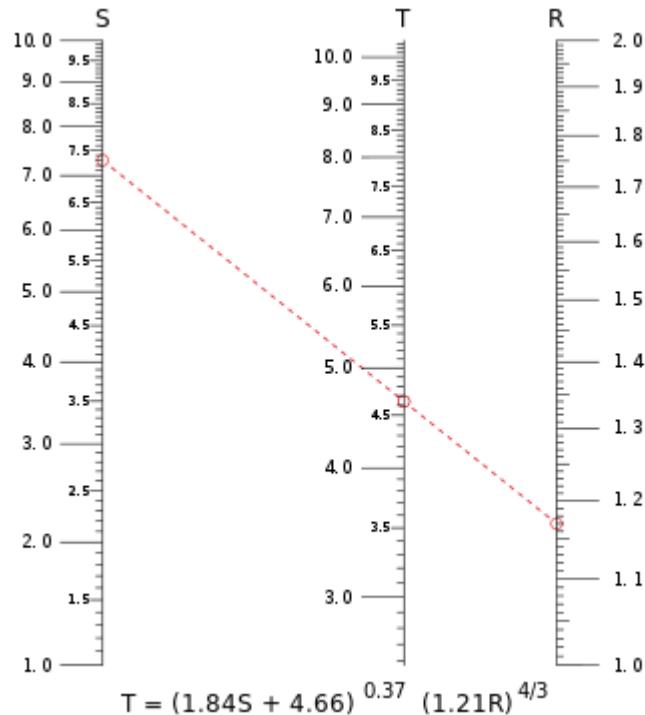
# Outliers



- Discretisation
- Capping / Censoring
- Truncation



# • Variable Magnitude

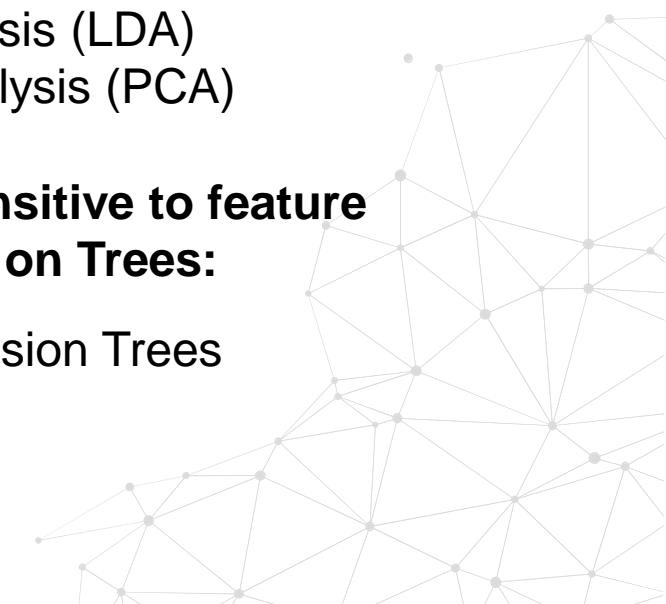


**The machine learning models affected by the magnitude of the feature:**

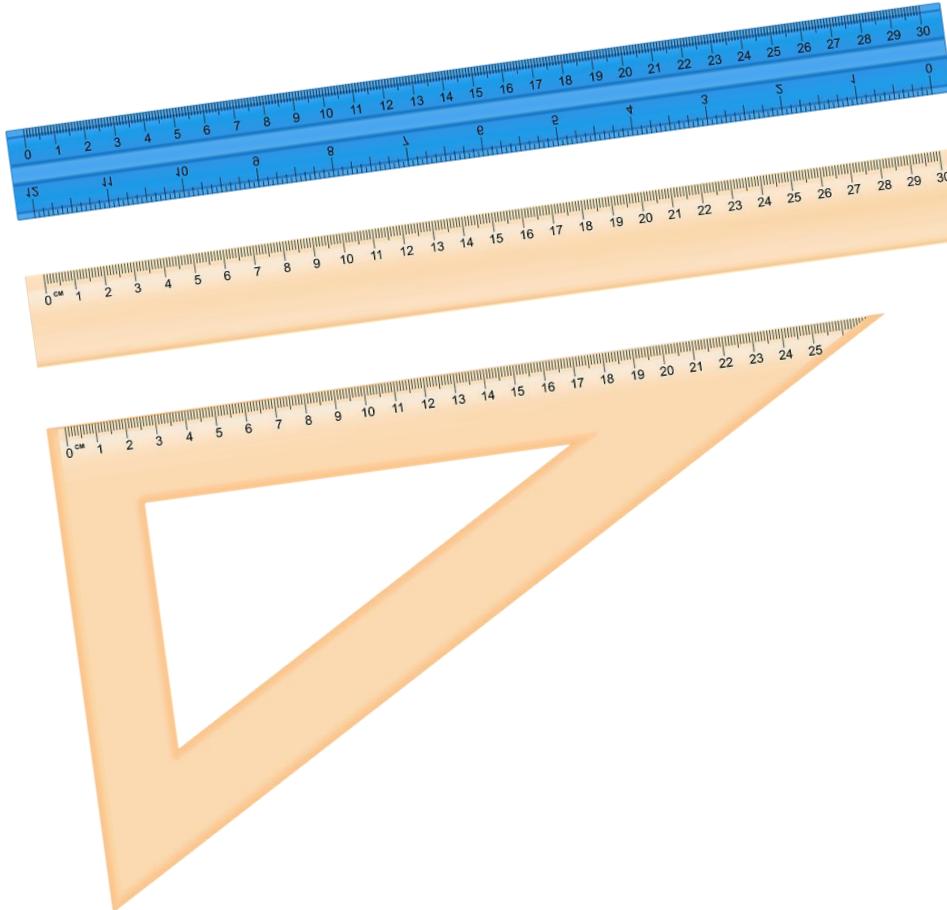
- Linear and Logistic Regression
- Neural Networks
- Support Vector Machines
- KNN
- K-means clustering
- Linear Discriminant Analysis (LDA)
- Principal Component Analysis (PCA)

**Machine learning models insensitive to feature magnitude are the ones based on Trees:**

- Classification and Regression Trees
- Random Forests
- Gradient Boosted Trees

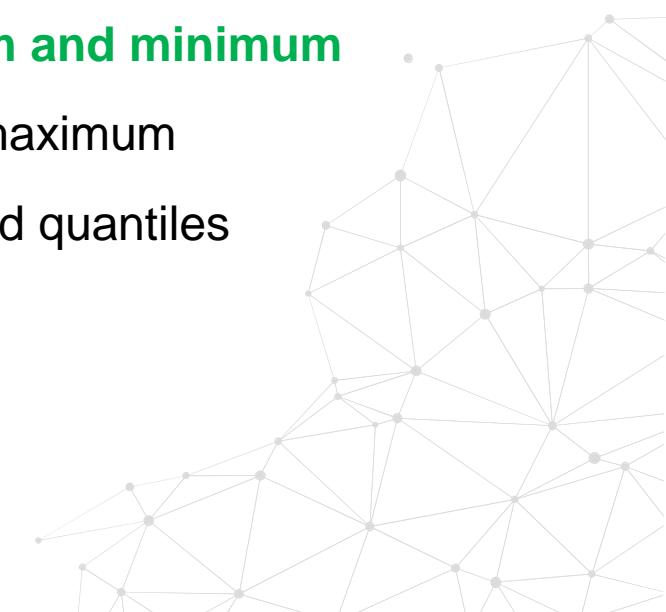


# • Feature scaling methods



## Scaling methods

- **Standardisation**
- Mean normalisation
- **Scaling to maximum and minimum**
- Scaling to absolute maximum
- Scaling to median and quantiles
- Scaling to unit norm



# Datetime Variables



- Day, Month, semester, year
- Hour, min, sec
- Elapsed Time
  - Time between transactions
  - Age



# Text

An insurance claim  
A formal request to an insurer for payment based on the terms of an insurance policy.  
Insurance claims are requests by policyholders for payment from their insurance company.

- Characters, words, unique words
- Lexical diversity
- Sentences, paragraphs
- Bag of Words
- TFIDF



# Transactions and Time Series



Aggregate data

- Number of payments in last 3, 6, 12 months
- Time since last transaction
- Total spending in last month



# Geo Data



- Distances



# • Feature Combination



- **Ratio:** Total debt with income → Debt to income ratio
- **Sum:** Debt in different credit cards → total debt
- **Subtraction:** Income without expenses → disposable income





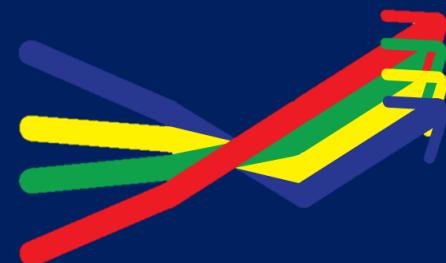
# Thank you

[www.trainindata.com](http://www.trainindata.com)



# Machine Learning Model Pipeline

## Feature Selection



# Why Do We Select Features?

- Simple models are easier to interpret
- Shorter training times
- Enhanced generalisation by reducing overfitting
- Easier to implement by software developers → Model production
- Reduced risk of data errors during model use
- Data redundancy

# Reducing features for model deployment

- Smaller json messages sent over to the model
  - Json messages contain only the necessary variables / inputs
- Less lines of code for error handling
  - Error handlers need to be written for each variable / input
- Less information to log
- Less feature engineering code

# Variable Redundancy



## Constant variables

Only 1 value per variable



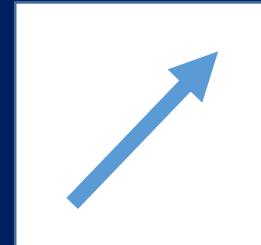
## Quasi – constant Variables

> 99% of observations show same value



## Duplication

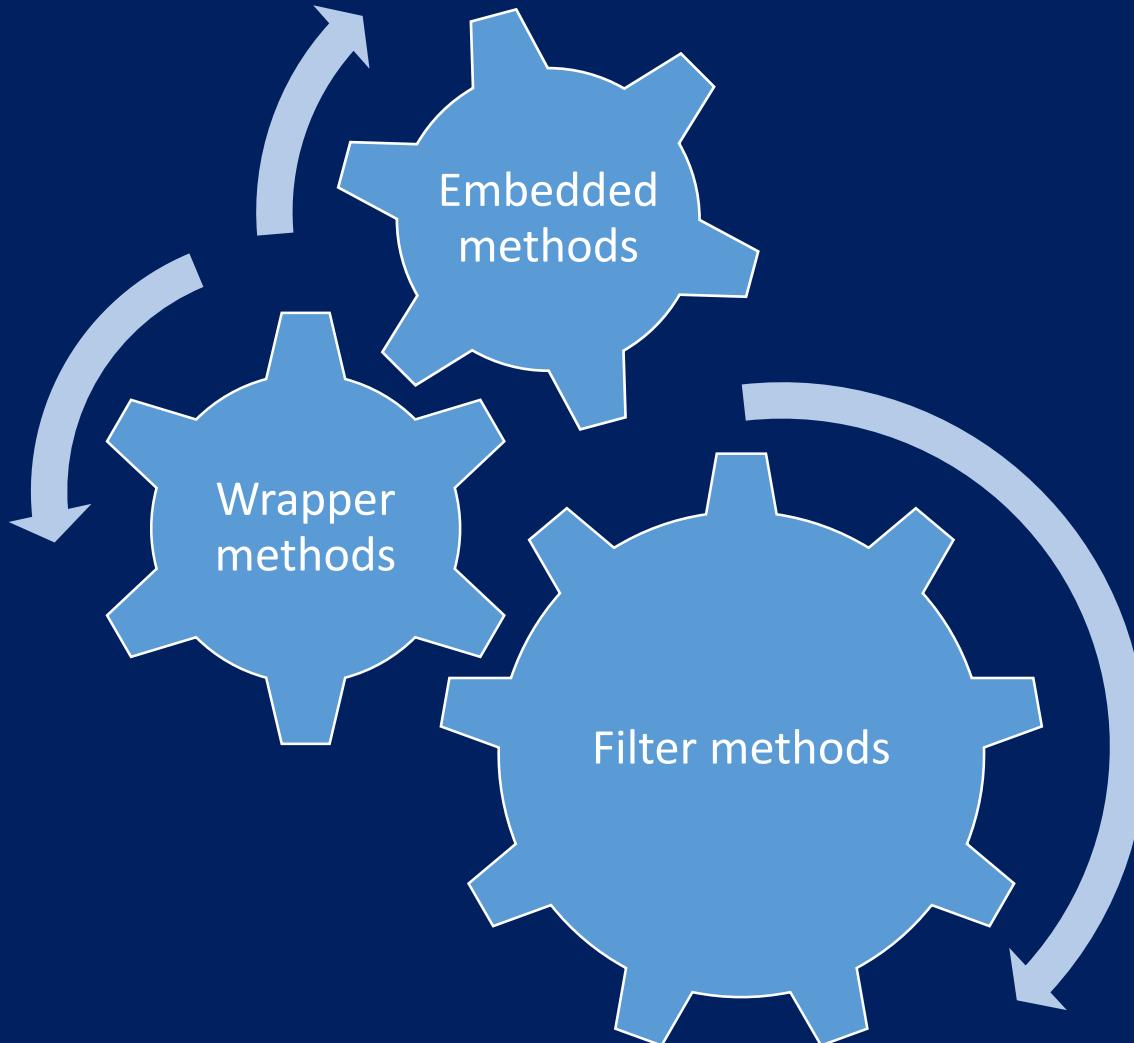
Same variable multiple times in the dataset



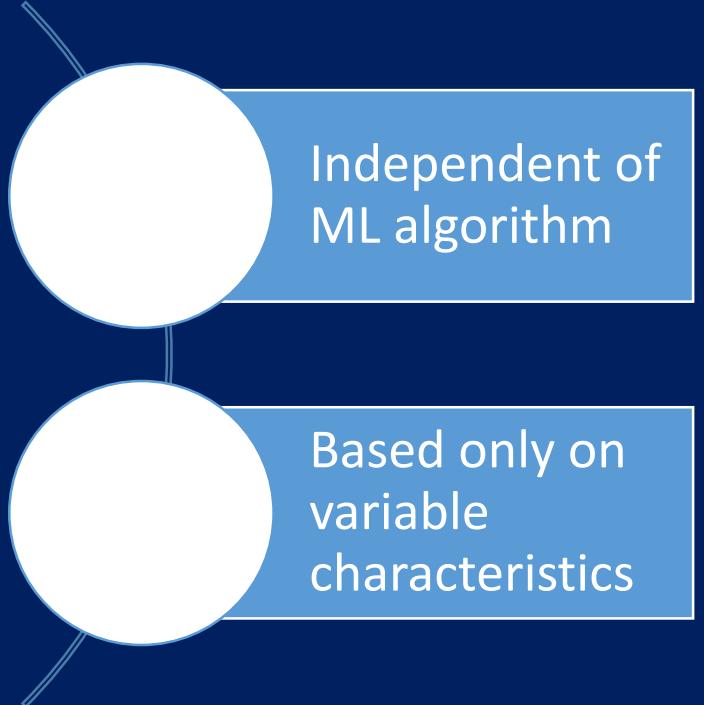
## Correlation

Correlated variables provide the same information

# Feature Selection Methods



# Filter methods



## Pros

Quick feature removal

Model agnostic

Fast computation

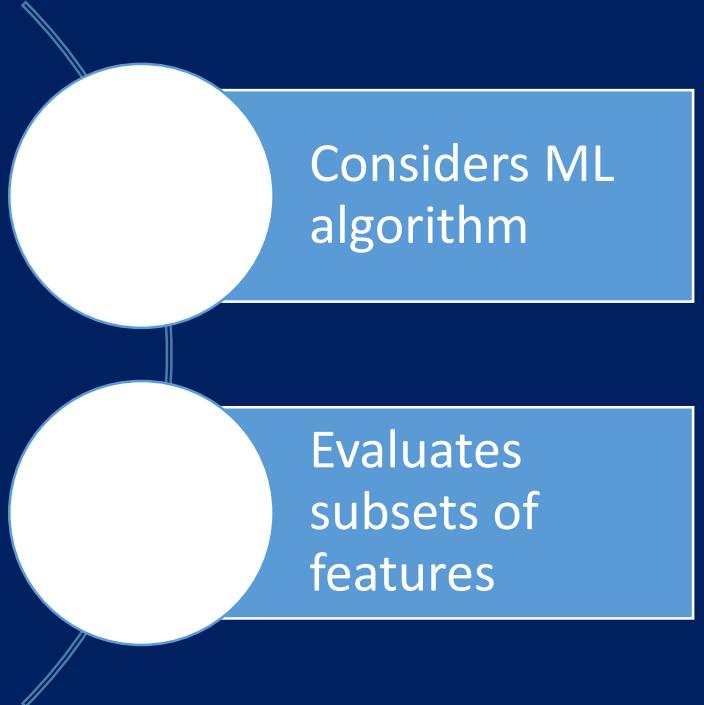
## Cons

Does not capture redundancy

Does not capture feature interaction

Poor model performance

# Wrapper methods



## Pros

Considers feature interaction

Best performance

Best feature subset for a given algorithm

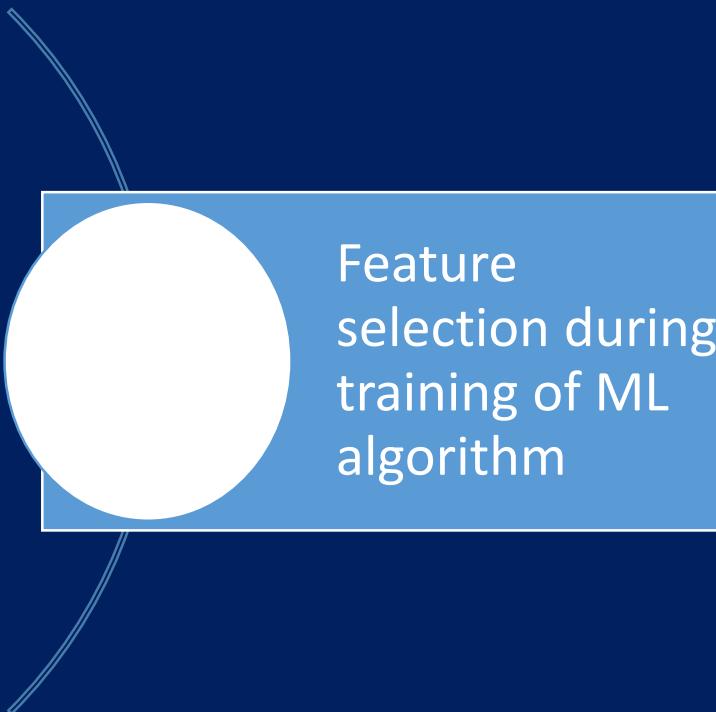
## Cons

Not model agnostic

Computation expensive

Often impracticable

# Embedded methods



Pros

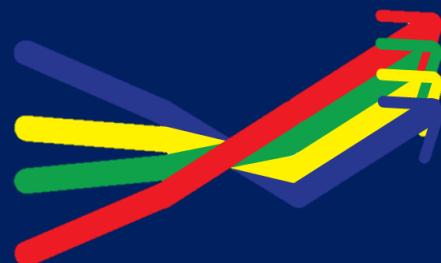
- Good model performance
- Capture feature interaction
- Better than filter
- Faster than wrapper

Cons

- Not model agnostic

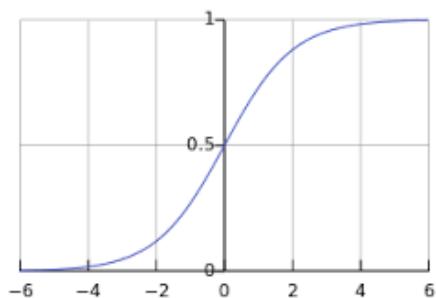
# Machine Learning Model Pipeline

## Model Building



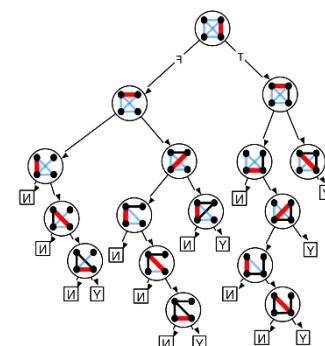
# Machine Learning Model Building

DATA

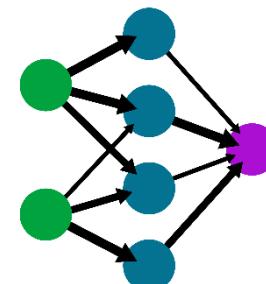


**Linear Models**  
Logistic Regression  
MARS

Prediction

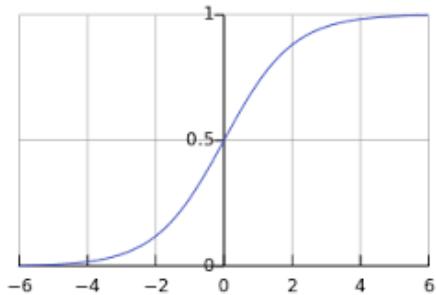


**Tree Models**  
Random Forests  
Gradient Boosted Trees

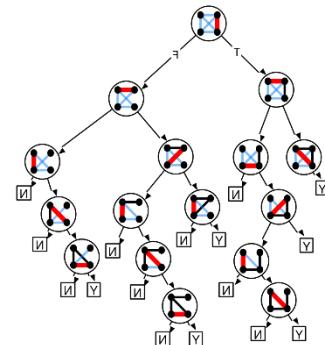


**Neural Networks**

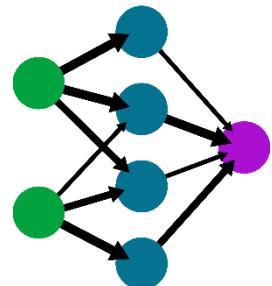
# Machine Learning Model - Performance



**Linear Models**  
Logistic Regression  
MARS



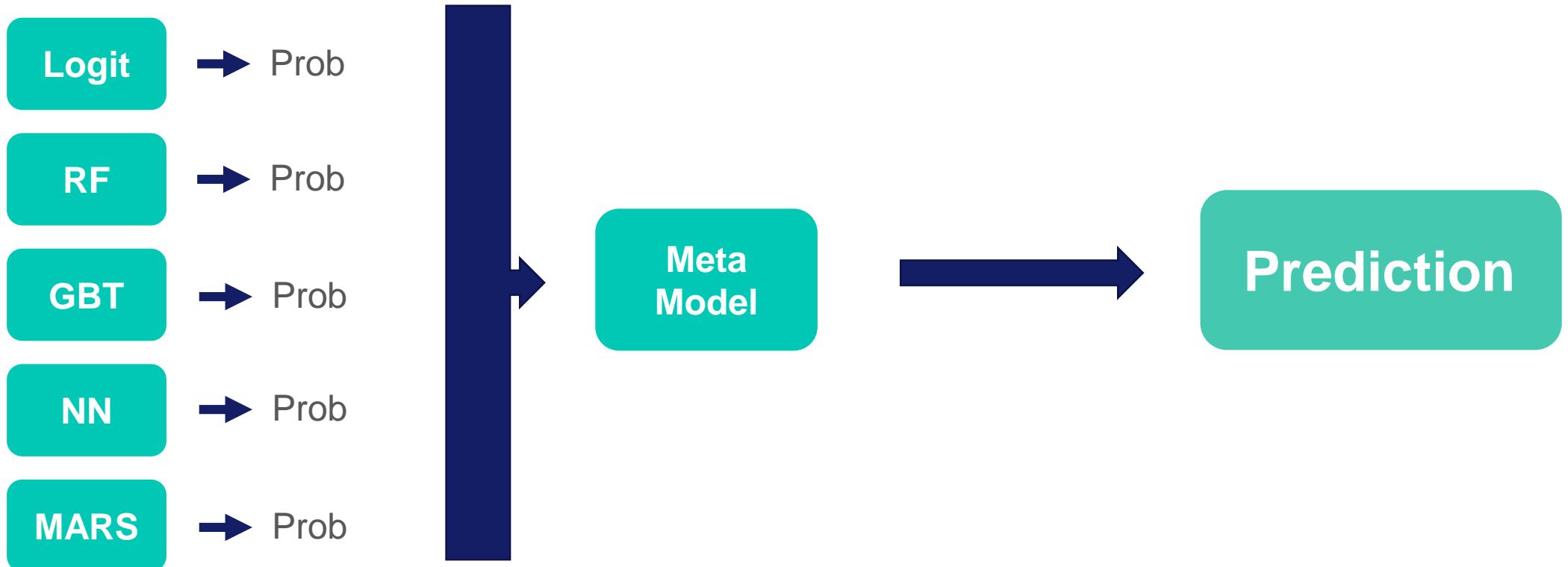
**Tree Models**  
Random Forests  
Gradient Boosted Trees



**Neural Networks**

- **ROC-AUC**
  - For each probability value:
    - How many times the model made a good assessment
    - How many times the model made a wrong assessment.
- **Accuracy**
- **MSE, RMSE, etc**

# Model Stacking – Meta Ensembling

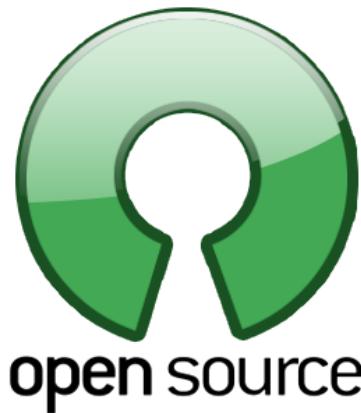


# Model Deployment





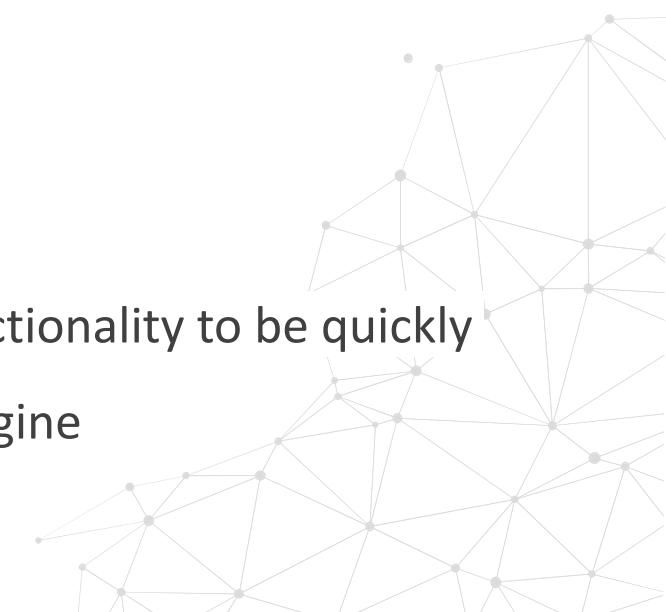
# Streamlining Machine Learning Pipelines with Open Source



# Scikit-Learn



- Solid implementation of a wide range of machine learning algorithms and data transformations
- Clean, uniform, and streamlined API.
- Most algorithms follow the same functionality → implementing new algos is super easy
  - Transformers
  - Estimators
  - Pipeline
- Complete online documentation, with some theory and examples
- Well established in the community → new packages follow Scikit-learn functionality to be quickly adopted by end users, e.g., Keras, MLXtend, category-encoders, Feature-engine



# Scikit-Learn Estimators

**Estimator** - A class with fit() and predict() methods.

It fits and predicts.

Any ML algorithm like Lasso, Decision trees, SVMs, are coded as estimators within Scikit-Learn.

```
class Estimator(object):  
  
    def fit(self, X, y=None):  
        """  
        Fits the estimator to data.  
        """  
        return self  
  
    def predict(self, X):  
        """  
        Compute the predictions  
        """  
        return predictions
```

# Scikit-Learn Transformers

**Transformers** - class that have fit() and transform() methods.

It transforms data.

- Scalers
- Feature selectors
- Encoders
- Imputers
- Discretizers
- Transformers

```
class Transformer(object):  
  
    def fit(self, X, y=None):  
        """  
        Learn the parameters to  
        engineer the features  
        """  
  
    def transform(X):  
        """  
        Transforms the input data  
        """  
        return X_transformed
```

# Scikit-Learn Pipeline

**Pipeline** - class that allows to run transformers and estimators in sequence.

- Most steps are Transformers
- Last step can be an Estimator

```
class Pipeline(Transformer):  
  
    @property  
    def name_steps(self):  
        """Sequence of transformers  
        ...  
        return self.steps  
  
    @property  
    def _final_estimator(self):  
        """  
        Estimator  
        ...  
        return self.steps[-1]
```

# Scikit-Learn Pipeline in action

[Here](#) is a good example of Pipeline usage. Pipeline gives you a single interface for all 3 steps of transformation and resulting estimator. It encapsulates transformers and predictors inside, and now you can do something like:

```
1 vect = CountVectorizer()
2 tfidf = TfidfTransformer()
3 clf = SGDClassifier()
4
5 vX = vect.fit_transform(Xtrain)
6 tfidfX = tfidf.fit_transform(vX)
7 predicted = clf.fit_predict(tfidfX)
8
9 # Now evaluate all steps on test set
10 vX = vect.transform(Xtest)
11 tfidfX = tfidf.transform(vX)
12 predicted = clf.predict(tfidfX)
```

With just:

```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier()),
])
predicted = pipeline.fit(Xtrain).predict(Xtrain)
# Now evaluate all steps on test set
predicted = pipeline.predict(Xtest)
```

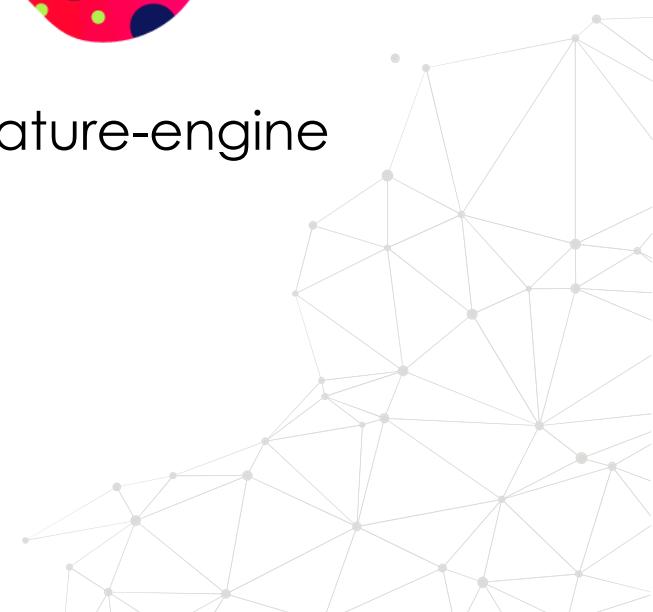
[Taken from stackoverflow](#)



# Open-source for Feature Engineering



Feature-engine



# Open-source for Feature Selection



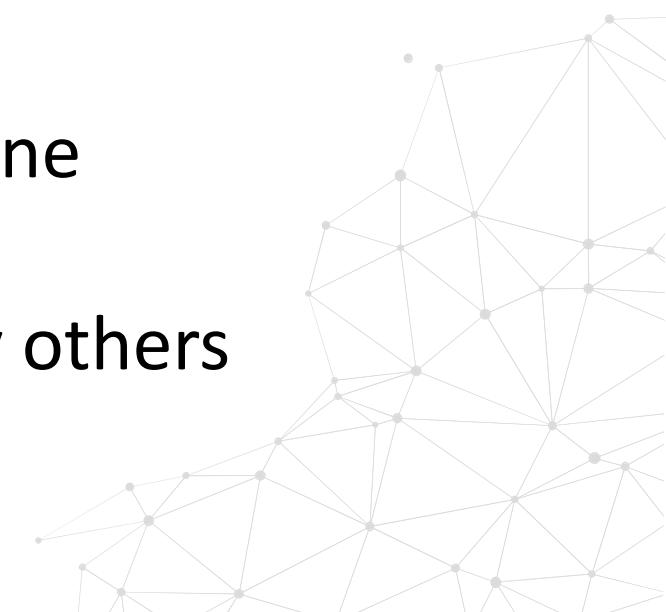
Feature-engine



# • Open-source for Model Training



- Py-earth
- xgboost
- Lasagne
- Many others





# Thank you

[www.trainindata.com](http://www.trainindata.com)





# Open Source for Feature Engineering



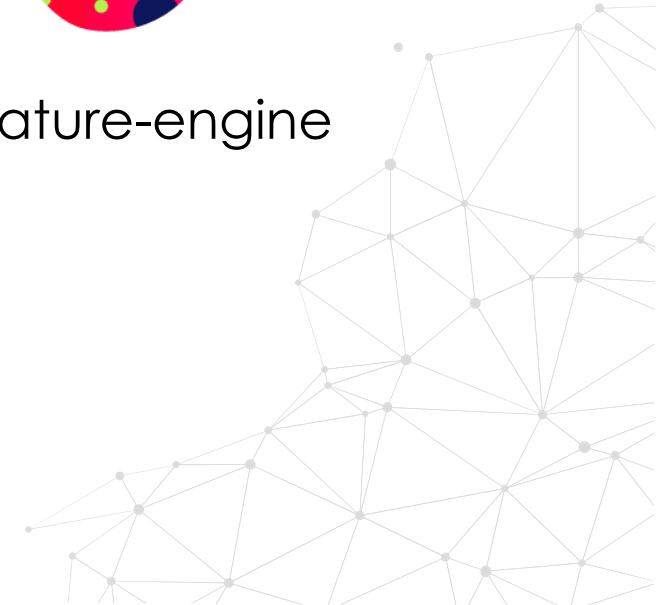
Category Encoders



- Open-source for Feature engineering



Feature-engine



# Scikit-Learn Transformers



- Missing Data Imputation
  - SimpleImputer
  - IterativeImputer
- Categorical Variable Encoding
  - OneHotEncoder
  - OrdinalEncoder
- Scalers
  - StandardScaler
  - MinMaxScaler
  - RobustScaler
  - A few others
- Discretisation
  - KBinsDiscretizer
- Variable Transformation
  - PowerTransformer
  - FunctionTransformer
- Variable Combination
  - Polynomial Features
- Text
  - Word Count
  - TFiDF



# Feature Engine Transformers

## Discretisation methods

- EqualFrequencyDiscretiser
- EqualWidthDiscretiser
- DecisionTreeDiscretiser
- ArbitraryDiscretiser

## Variable Transformation methods

- LogTransformer
- ReciprocalTransformer
- PowerTransformer
- BoxCoxTransformer
- YeoJohnsonTransformer

## Scikit-learn Wrapper:

- SklearnTransformerWrapper

## Variable Combinations:

- MathematicalCombination
- CombineWithReferenceFeature

## Imputing Methods

- MeanMedianImputer
- RandomSampleImputer
- EndTailImputer
- AddMissingIndicator
- CategoricalImputer
- ArbitraryNumberImputer
- DropMissingData

## Encoding Methods

- OneHotEncoder
- OrdinalEncoder
- CountFrequencyEncoder
- MeanEncoder
- WoEEncoder
- PRatioEncoder
- RareLabelEncoder
- DecisionTreeEncoder

## Outlier Handling methods

- Winsorizer
- ArbitraryOutlierCapper
- OutlierTrimmer



# Category Encoders

```
import category_encoders as ce

encoder = ce.BackwardDifferenceEncoder(cols=[...])
encoder = ce.BaseNEncoder(cols=[...])
encoder = ce.BinaryEncoder(cols=[...])
encoder = ce.CatBoostEncoder(cols=[...])
encoder = ce.HashingEncoder(cols=[...])
encoder = ce.HelmertEncoder(cols=[...])
encoder = ce.JamesSteinEncoder(cols=[...])
encoder = ce.LeaveOneOutEncoder(cols=[...])
encoder = ce.MEstimateEncoder(cols=[...])
encoder = ce.OneHotEncoder(cols=[...])
encoder = ce.OrdinalEncoder(cols=[...])
encoder = ce.SumEncoder(cols=[...])
encoder = ce.PolynomialEncoder(cols=[...])
encoder = ce.TargetEncoder(cols=[...])
encoder = ce.WOEEncoder(cols=[...])
```

The sidebar on the left contains a navigation bar with back, forward, and search icons, followed by the title "Category Encoders" and the "latest" version. Below the title is a search bar labeled "Search docs". A vertical list of encoder types is provided:

- Backward Difference Coding
- BaseN
- Binary
- CatBoost Encoder
- Hashing
- Helmert Coding
- James-Stein Encoder
- Leave One Out
- M-estimate
- One Hot
- Ordinal
- Polynomial Coding
- Sum Coding
- Target Encoder
- Weight of Evidence

The main content area shows the "Category Encoders" page. At the top right is the URL "contrib.scikit-learn.org/categorical-encoding/" and a "View page source" link. Below the header, there's a breadcrumb trail "Docs » Category Encoders". The main content starts with a section titled "Category Encoders" which describes the library as a set of scikit-learn-style transformers for encoding categorical variables into numeric. It lists several useful properties:

- First-class support for pandas dataframes as an input (and optionally as output)
- Can explicitly configure which columns in the data are encoded by name or index, or infer non-numeric columns regardless of input type
- Can drop any columns with very low variance based on training set optionally
- Portability: train a transformer on data, pickle it, reuse it later and get the same thing out.
- Full compatibility with sklearn pipelines, input an array-like dataset like any other transformer

Below this is a "Usage" section with instructions for installation:

install as:

```
pip install category_encoders
```

or

```
conda install -c conda-forge category_encoders
```

# Pipeline with Feature-engine



```
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline as pipe
from sklearn.preprocessing import MinMaxScaler

from feature_engine import categorical_encoders as ce
from feature_engine import discretisers as dsc
from feature_engine import missing_data_imputers as mdi

# Load dataset
data = pd.read_csv('houseprice.csv')

# drop some variables
data.drop(labels=['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'Id'], axis=1, inplace=True)

# categorical encoders work only with object type variables
data[discrete] = data[discrete].astype('O')

# separate into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data.drop(labels=['SalePrice'], axis=1),
                                                    data.SalePrice,
                                                    test_size=0.1,
                                                    random_state=0)

# set up the pipeline
price_pipe = pipe([
    # add a binary variable to indicate missing information for the 2 variables below
    ('continuous_var_imputer', mdi.AddNaNBinaryImputer(variables = ['LotFrontage'])),

    # replace NA by the median in the 2 variables below, they are numerical
    ('continuous_var_median_imputer', mdi.MeanMedianImputer(imputation_method='median', variables = ['LotFrontage', 'MasVnrArea'])),

    # replace NA by adding the label "Missing" in categorical variables
    ('categorical_imputer', mdi.CategoricalVariableImputer(variables = categorical)),

    # discretise numerical variables using trees
    ('numerical_tree_discretiser', dsc.DecisionTreeDiscretiser(cv = 3, scoring='neg_mean_squared_error')),

    # remove rare labels in categorical and discrete variables
    ('rare_label_encoder', ce.RareLabelCategoricalEncoder(tol = 0.03, n_categories=1, variables = categorical)),

    # encode categorical and discrete variables using the target mean
    ('categorical_encoder', ce.MeanCategoricalEncoder(variables = categorical+discrete)),

    # scale features
    ('scaler', MinMaxScaler()),

    # Lasso
    ('lasso', Lasso(random_state=2909, alpha=0.005))
])

# train feature engineering transformers and Lasso
price_pipe.fit(X_train, np.log(y_train))

# predict
pred_train = price_pipe.predict(X_train)
pred_test = price_pipe.predict(X_test)
```



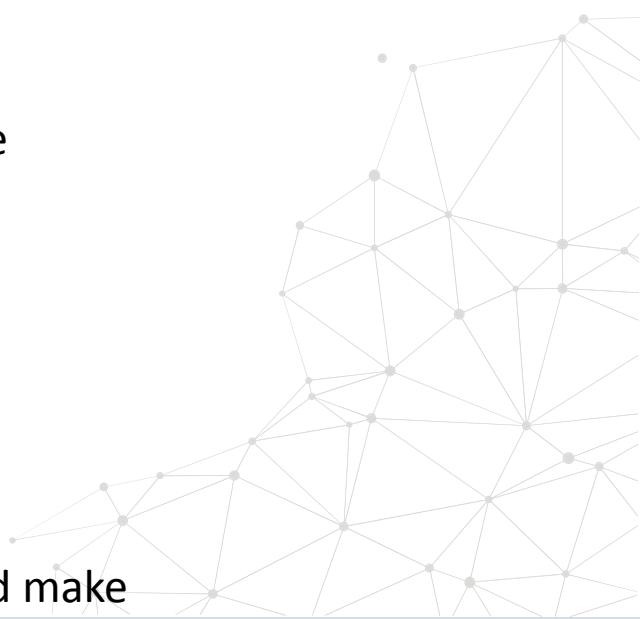
Import predefined  
transformers



Accommodate  
transformers in the  
pipeline



Fit the pipeline and make  
predictions



# • Pipeline



Category Encoders

```
► # Creamos pipeline
titanic_pipe = Pipeline([
    # imputación de datos ausentes - sección 4
    ('imputer_num',
     mdi.ArbitraryNumberImputer(arbitrary_number=-1,
                                  variables=['age', 'fare', 'cabin_num'])),
    ('imputer_cat',
     mdi.CategoricalVariableImputer(variables=['embarked', 'cabin_cat'])),
    # codificación de variables categóricas - sección 6
    ('encoder_rare_label',
     ce.RareLabelCategoricalEncoder(tol=0.01,
                                    n_categories=6,
                                    variables=['cabin_cat'])),
    ('categorical_encoder',
     ce.OrdinalCategoricalEncoder(encoding_method='ordered',
                                   variables=['cabin_cat', 'sex', 'embarked'])),
    # máquina de potenciación de gradiente
    ('gbm', GradientBoostingClassifier(random_state=0))
])
```

# • Pipeline



Category Encoders

```
# train pipeline  
price_pipe.fit(X_train, y_train)  
  
# transform data  
price_pipe.predict(X_train)  
price_pipe.predict(X_test)  
  
price_pipe.predict(live_data)
```



# • Pipeline



Category Encoders

```
# train pipeline  
price_pipe.fit(X_train, y_train)  
  
# transform data  
price_pipe.predict(X_train)  
price_pipe.predict(X_test)  
  
price_pipe.predict(live_data)
```



Python Pickle

Python Pickle

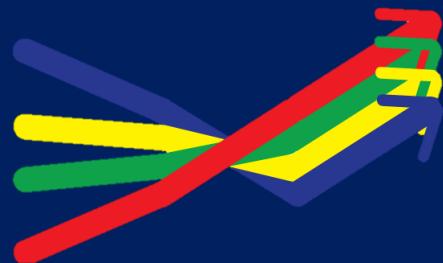


# Thank you

[www.trainindata.com](http://www.trainindata.com)



# Object Oriented Programming for Machine Learning



# Learn and transform, learn and predict

The different steps of the Machine Learning pipeline:

- Learn parameters from data
- Use those parameters to make transformations or predictions

# Procedural Programming in ML

Code:

- Learn the parameters
- Make the transformations
- Make the predictions

Data:

- Store the parameters
- Mean values, regression coefficients, etc.

# Procedural Programming: Hard-coded values



```
15
14 # encoding parameters
16 FREQUENT_LABELS = {
17     'MSZoning': ['FV', 'RH', 'RL', 'RM'],
18     'Neighborhood': ['Blmgtn', 'BrDale', 'BrkSide', 'ClearCr', 'CollgCr',
19                      'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV',
20                      'Mitchel', 'NAmes', 'NWAmes', 'NoRidge', 'NridgHt',
21                      'OldTown', 'SWISU', 'Sawyer', 'SawyerW', 'Somerst',
22                      'StoneBr', 'Timber'],
23     'RoofStyle': ['Gable', 'Hip'],
24     'MasVnrType': ['BrkFace', 'None', 'Stone'],
25     'BsmtQual': ['Ex', 'Fa', 'Gd', 'Missing', 'TA'],
26     'BsmtExposure': ['Av', 'Gd', 'Missing', 'Mn', 'No'],
27     'HeatingQC': ['Ex', 'Fa', 'Gd', 'TA'],
28     'CentralAir': ['N', 'Y'],
29     'KitchenQual': ['Ex', 'Fa', 'Gd', 'TA'],
30     'FireplaceQu': ['Ex', 'Fa', 'Gd', 'Missing', 'Po', 'TA'],
31     'GarageType': ['Attchd', 'Basment', 'BuiltIn', 'Detchd', 'Missing'],
32     'GarageFinish': ['Fin', 'Missing', 'RFn', 'Unf'],
33     'PavedDrive': ['N', 'P', 'Y']}
34
35
36 ENCODING_MAPPINGS = {'MSZoning': {'Rare': 0, 'RM': 1, 'RH': 2, 'RL': 3, 'FV': 4},
37                       'Neighborhood': {'IDOTRR': 0, 'MeadowV': 1, 'BrDale': 2, 'Edwards': 3,
38                           'BrkSide': 4, 'OldTown': 5, 'Sawyer': 6, 'SWISU': 7, 'NAmes':
39                           'Mitchel': 9, 'SawyerW': 10, 'Rare': 11, 'NWAmes': 12, 'Gilber':
40                           'Blmgtn': 14, 'CollgCr': 15, 'Crawfor': 16, 'ClearCr': 17, 'S
41                           'Timber': 19, 'StoneBr': 20, 'NridgHt': 21, 'NoRidge': 22},
42
43     'RoofStyle': {'Gable': 0, 'Rare': 1, 'Hip': 2},
44     'MasVnrType': {'None': 0, 'Rare': 1, 'BrkFace': 2, 'Stone': 3},
45     'BsmtQual': {'Missing': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
46     'BsmtExposure': {'Missing': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4},
47     'HeatingQC': {'Rare': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
48     'CentralAir': {'N': 0, 'Y': 1},
49     'KitchenQual': {'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
50     'FireplaceQu': {'Po': 0, 'Missing': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
51     'GarageType': {'Missing': 0, 'Rare': 1, 'Detchd': 2, 'Basment': 3, 'Attchd': 4,
52     'GarageFinish': {'Missing': 0, 'Unf': 1, 'RFn': 2, 'Fin': 3},
53     'PavedDrive': {'N': 0, 'P': 1, 'Y': 2}}
54
55
56 # ===== FEATURE GROUPS ======
```

Line 1, Column 1      master [9]      Tab Size: 4

- 
- 
- Straightforward
  - Hard-code parameters
  - Save multiple objects or data structures

```
OUTPUT_SCALER_PATH = 'scaler.pkl'
OUTPUT_MODEL_PATH = 'lasso_regression.pkl'
```

# Object Oriented Programming - OOP

In **Object-oriented programming (OOP)** we write code in the form of “objects”.

This “objects” can store **data** and can also store instructions or procedures (**code**) to modify that data, or do something else, like obtaining predictions.

- Data ⇒ attributes, properties
- Code or Instructions ⇒ methods (procedures)

# OOP for Machine Learning

In **Object-oriented programming (OOP)** the “objects” can learn and store parameters

- Parameters get automatically refreshed every time model is re-trained
- No need of manual hard-coding
- Methods:
  - Fit: learns parameters
  - Transform: transforms data with the learned parameters
- Attributes: store the learn parameters

# A Class

Creating a Class in Python

```
class MeanImputer:  
    pass
```

# A Class - `__init__()`

The properties or parameters that the class takes whenever it is initialized, are indicated in the `__init__()` method.

The first parameters will always be a variable called `self`.

```
class MeanImputer:  
  
    def __init__(self, variables):  
        self.variables = variables
```

We can give any number of parameters to `__init__()`

# A Class - `__init__()`

The properties or parameters that the class takes whenever it is initialized, are indicated in the `__init__()` method.

The first parameters will always be a variable called `self`.

We can give any number of parameters to `__init__()`

```
class MeanImputer:  
  
    def __init__(self, variables):  
        self.variables = variables
```

```
>> my_imputer = MeanImputer(  
>>         variables = ['age', 'fare']  
>> )  
  
>> my_imputer.variables  
['age', 'fare']
```

# A Class - methods

**Methods** are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called `self`.

Our `fit()` method learns parameters

```
class MeanImputer:  
    def __init__(self, variables):  
        self.variables = variables  
  
    def fit(self, X, y=None):  
        self.imputer_dict_ =  
            X[self.variables].mean().to_dict()  
  
        return self
```

# A Class - methods

**Methods** are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called `self`.

Our `fit()` method learns parameters

```
>> my_imputer = MeanImputer(  
>>         variables = ['age', 'fare']  
>> )  
  
>> my_imputer.fit(my_data)  
>> my_imputer.imputer_dict_  
  
{'age': 39, 'fare': 100}
```

# A Class - methods

**Methods** are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called `self`.

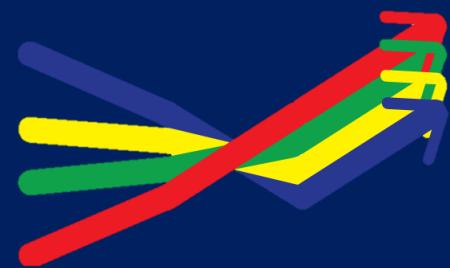
Our `fit()` method learns parameters

Our `transform()` method transforms data

```
class MeanImputer:  
    def __init__(self, variables):  
        self.variables = variables  
  
    def fit(self, X, y=None):  
        self.imputer_dict_ =  
            X[self.variables].mean().to_dict()  
        return self  
  
    def transform(self, X):  
        for x in self.variables:  
            X[x] = X[x].fillna(  
                self.imputer_dict_[x])  
        return X
```

# Object Oriented Programming

## - Inheritance



# Inheritance

**Inheritance** is the process by which one class takes on the **attributes** and **methods** of another.

# Parent Class

The properties or parameters that the class takes whenever it is initialized, are indicated in the `__init__()` method.

The first parameters will always be a variable called `self`.

We can give any number of parameters to `__init__()`

```
class TransformerMixin:  
  
    def fit_transform(self, X, y=None):  
        X = self.fit(X, y).transform(X)  
        return X
```

# Our MeanImputer – Child Class

**Inherits** the methods `fit_transform()` from the `TransformerMixin`

```
class MeanImputer(TransformerMixin):  
    def __init__(self, variables):  
        self.variables = variables  
  
    def fit(self, X, y=None):  
        self.imputer_dict_ =  
            X[self.variables].mean().to_dict()  
        return self  
  
    def transform(self, X):  
        for x in self.variables:  
            X[x] = X[x].fillna(  
                self.imputer_dict_[x])  
        return X
```

# Our MeanImputer – Child Class

**Inherits** the methods `fit_transform()` from the `TransformerMixin`

```
>> my_imputer = MeanImputer(  
>>         variables = ['age', 'fare']  
>> )  
  
>> data_t = my_imputer.fit_transform(my_data)  
>> data_t.head()
```

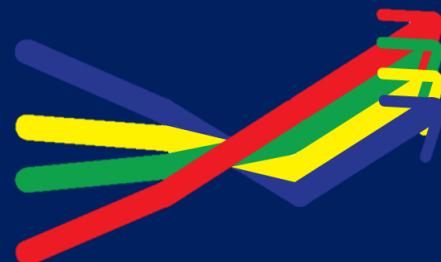
	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	0.083333	0.0	0.495064	0.0	0.0	0.0	0.666667
1	0.083333	0.0	0.499662	0.0	0.0	0.0	0.666667
2	0.083333	0.0	0.466207	0.0	0.0	0.0	0.666667
3	0.083333	0.0	0.485693	0.0	0.0	0.0	0.666667
4	0.083333	0.0	0.265271	0.0	0.0	0.0	0.666667

# Scikit-Learn API documentation

<https://scikit-learn.org/stable/modules/classes.html>

- [base.BaseEstimator](#)
- [base.TransformerMixin](#)

# Should feature selection be part of the Pipeline?



# Feature selection in the pipeline

If deploying for the first time,

Plus

- We don't have to hard code the predictive features

However,

- Need to deploy code to engineer **all features** in the dataset
- Error handling and unit testing for all the code to engineering features



# Feature selection in the pipeline

**What if we re-train our model frequently?**

## **Advantages**

- Can quickly retrain a model on the same input data
- No need to hard-code the new set of predictive features after each re-training

## **Disadvantages**

- Lack of data versatility
- No additional data can be fed through the pipeline

# Feature selection in the pipeline

**In summary,**

**Suitable:**

- Model build and refreshed on same data
- Model build and refreshed on smaller datasets

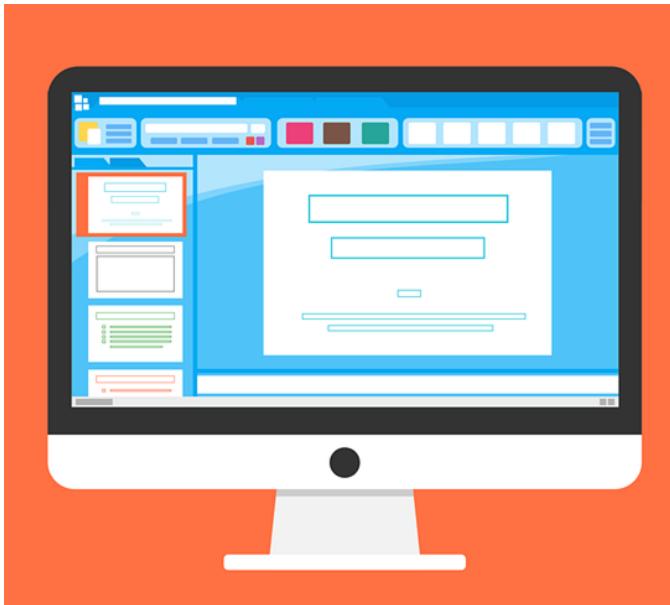
**Not suitable,**

- If model built using datasets with a high feature space
- If model constantly enriched with new data sources



# Production Code Introduction

# What do we mean by "Production Code"?



- Let's dig into the terminology to avoid confusion...



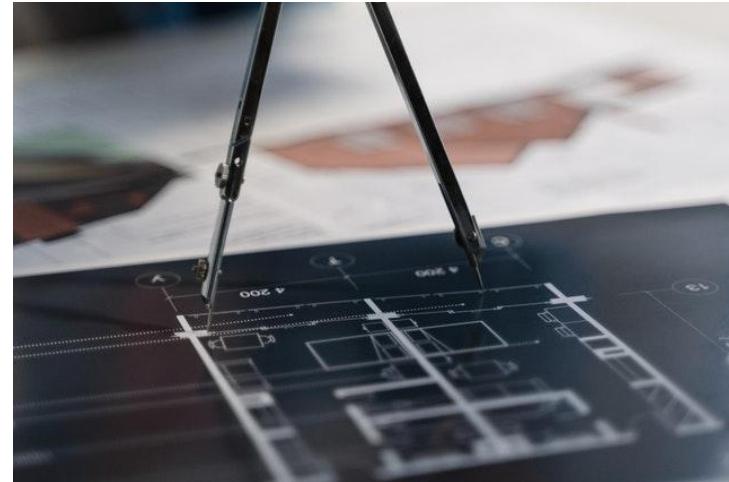


# Purpose

Production code is designed to be deployed to end users.

# Production Code Considerations

- Testability & Maintainability



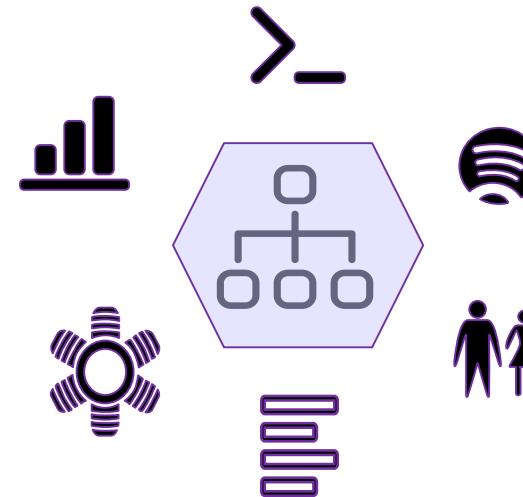
# Production Code Considerations

- Scalability & Performance



# Production Code Considerations

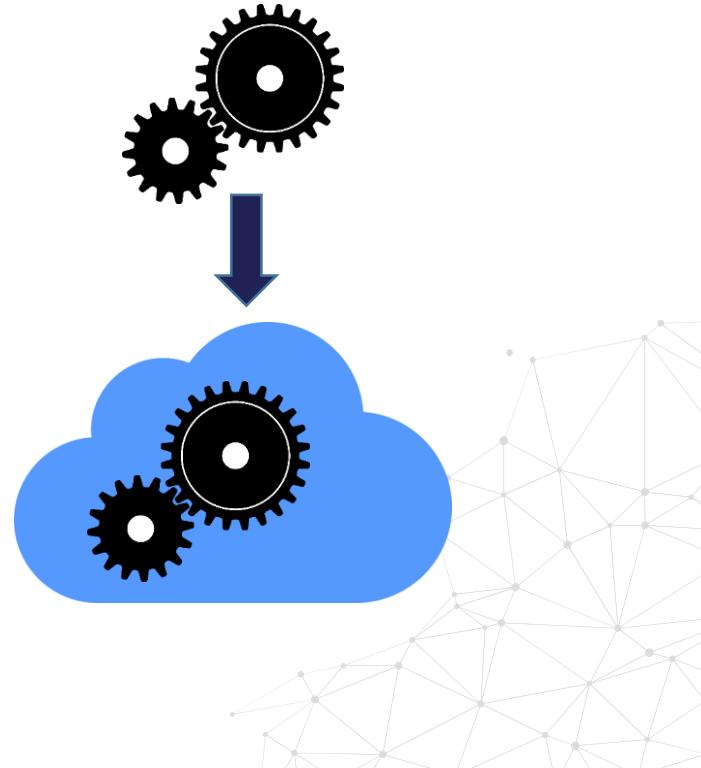
- Reproducibility



# Python Packages

A **module** is a file which contains various Python functions and global variables. It is just a file with a `.py` extension which has python executable code.

A **package** is a collection of modules.



# Package Structure

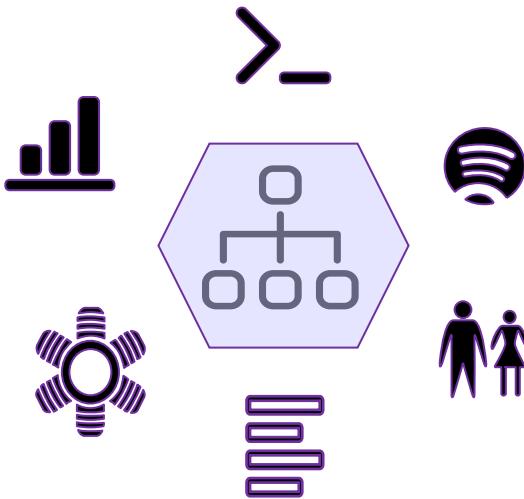
```
parent/
    └── regression_model/
        └── config/
        └── datasets/
        └── processing/
        └── trained_models/
        └── config.yml
        └── pipeline.py
        └── predict.py
        └── train_pipeline.py
        └── VERSION
    └── requirements/
        └── requirements.txt
        └── test_requirements.txt
    └── setup.py
    └── MANIFEST.in
    └── pyproject.toml
    └── tox.ini
    └── tests/
```





# Architecture of our ML API + Implications

# ML System Architectures



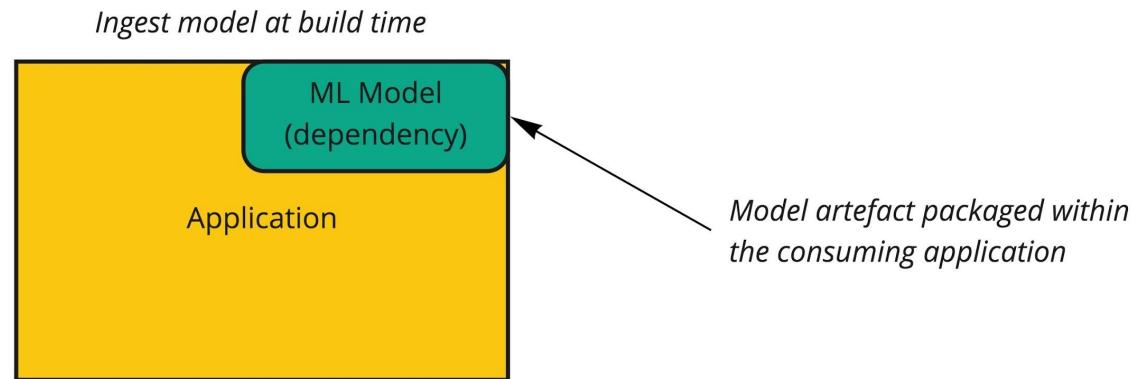
1. Model embedded in application
2. Served via a dedicated service
3. Model published as data (streaming)
4. Batch prediction (offline process)



# Architecture 1: Embedded

**Pre-Trained:** Yes

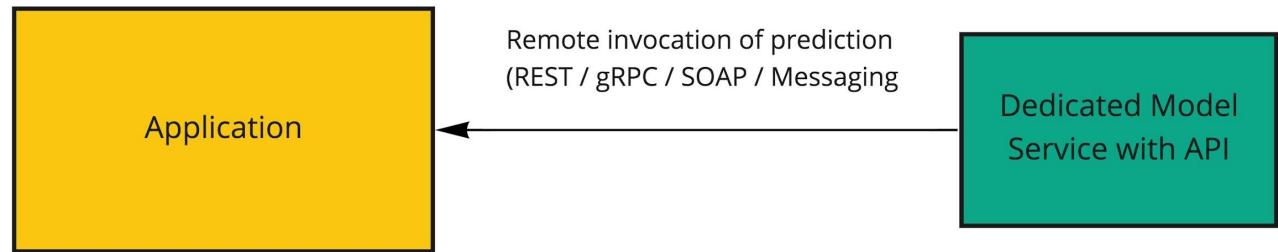
**Predict-on-the-fly:** Yes



# Architecture 2: Dedicated Model API

Pre-Trained: Yes

Predict-on-the-fly: Yes

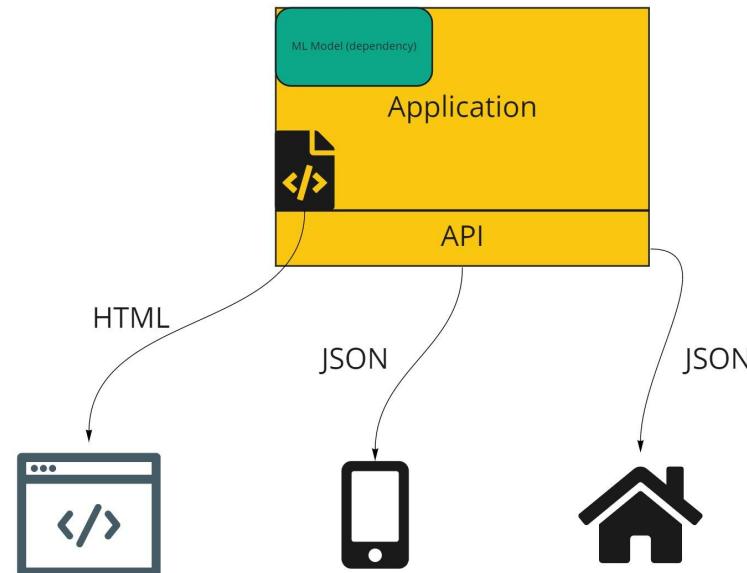


*Model is wrapped in a service that can be deployed independently*



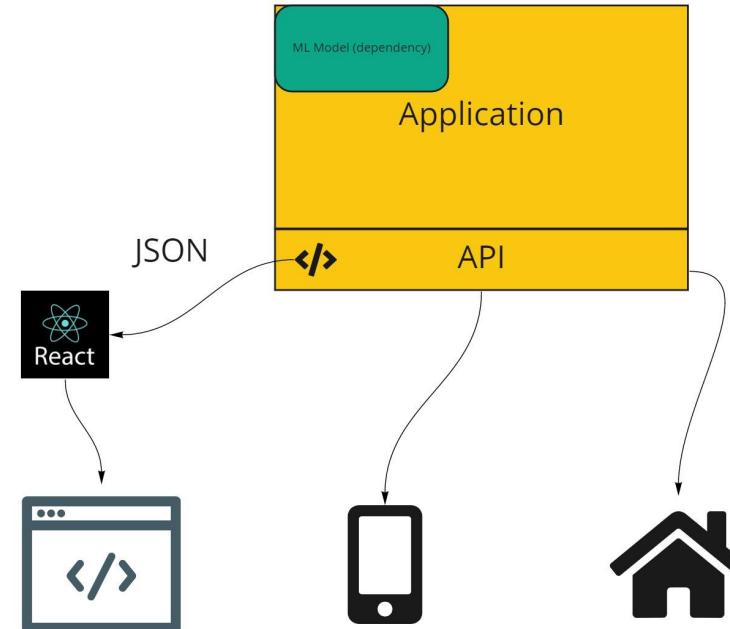
# How Can Our API Be Consumed?

- Web browsers (HTML)
- Mobile Devices
- IOT
- Other applications



# Modern Frontend Approaches

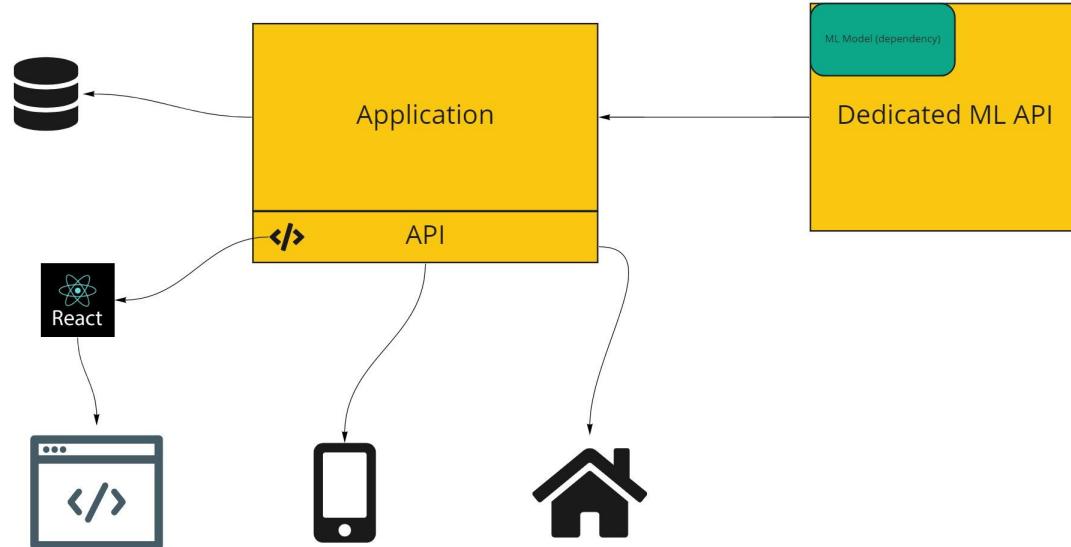
**JS Frameworks like  
ReactJS, Vue  
and AngularJS**



# Dedicated ML API + Microservices

Refer to section 3  
for a discussion of  
the trade-offs

Our example project  
is not technically  
"dedicated" - but it's  
close.





# Introducing FastAPI

# Web Frameworks

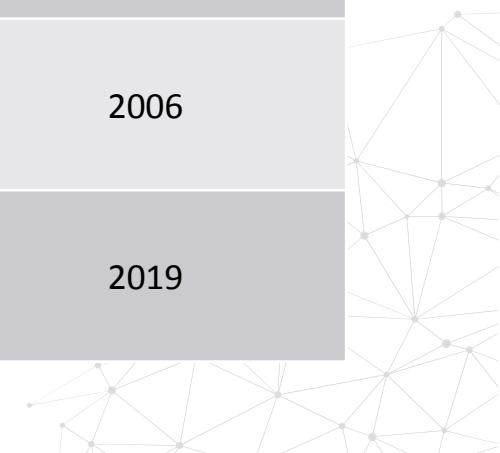
Provide tools to handle & automate standard tasks when building web applications

- Session management
- Templating
- Database Access
- Much more!



# Python Web Frameworks

	Github Stars	Github Forks	Launch Year
Flask	55.3k	14.3k	2011
Django	57.5k	24.5k	2006
FastAPI	31.1k	2.2k	2019



# • Features of FastAPI



- It's Fast – leverages Python async capabilities
- Validation with type hints and Pydantic
- Automatic Documentation
- Dependency Injection
- Much more!

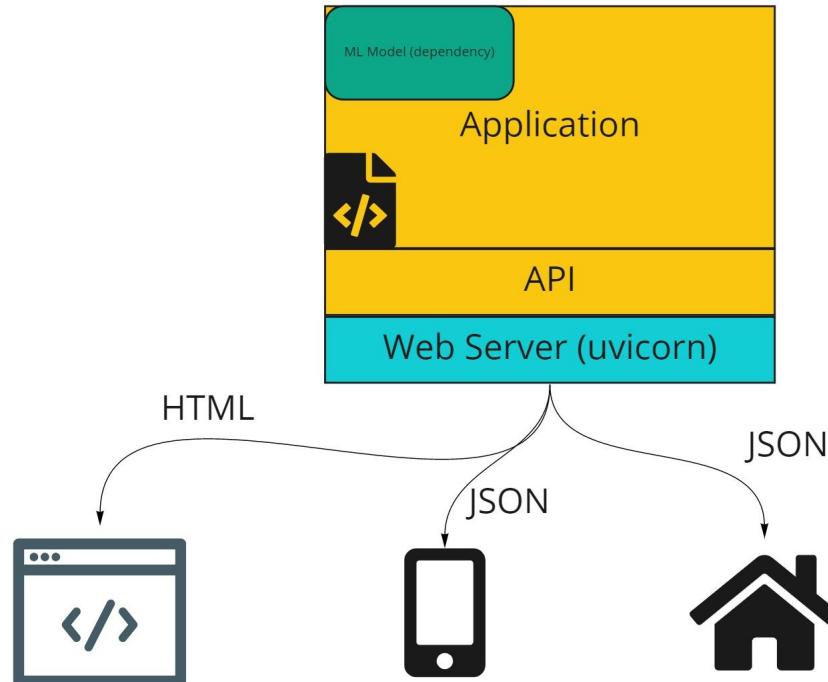




# Uvicorn ASGI Web Server

# • Production Components: Web Server

- For a production deployment, we require a web server
- Although we tend to use the terms interchangeably, technically the server deals with handling incoming requests and outgoing responses
- A server implements a "server gateway interface"



# WSGI & ASGI

- Historically, Python web frameworks have used the Web Server Gateway Interface (WSGI) introduced in PEP 333 (and revised in PEP 3333)
- With the introduction of Python's `asyncio` library it was possible to go beyond the constraints of WSGI (which could not be upgraded without being backwards incompatible). Hence the Asynchronous Server Gateway Interface (ASGI) was born.
- Uvicorn is an implementation of this interface.





# Heroku as our Platform as a Service? (PaaS)

# What is a Platform as a Service (PaaS)?

“A cloud-based service that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.”

# Understanding the Range of Possibilities

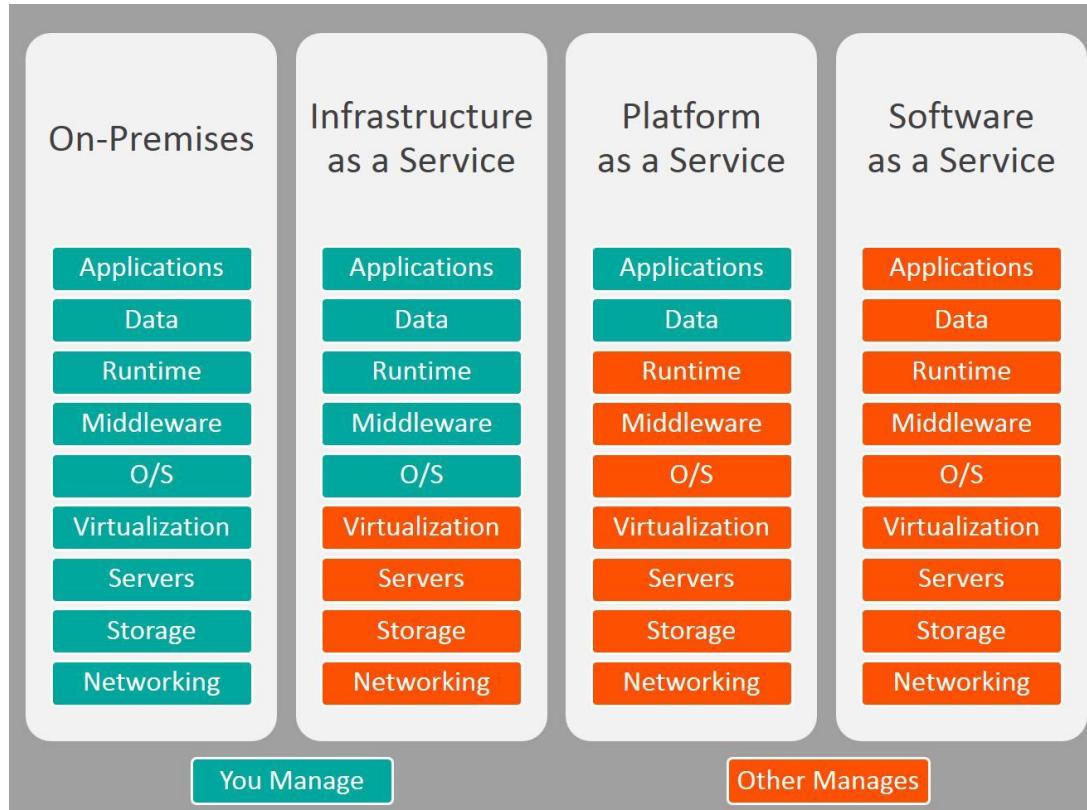


Image copyright:  
<https://www.bmc.com>

# PaaS Pros and Cons

Pros	Cons
Simple to setup, maintain and deploy	Hard / impossible to scale to a very large size
Easy to scale to moderate size	Tends to be more expensive than IaaS
Allows developers to focus on apps	Vulnerable to PaaS downtime
Easy creation of dev/test environments	Limitations on configuration

# PaaS Providers

- AWS Elastic Beanstalk
- Windows Azure
- Heroku
- Force.com
- Google App Engine
- Apache Stratos
- OpenShift
- PythonAnywhere
- ...and many more.

# Why Heroku in this Course?



- Heroku is very easy to use
- We can use one Heroku Dyno for free (ideal for teaching).
- Nice 3rd Party Add on options
- Works with Docker
- Great documentation
- Supports multiple languages

# What is CI/CD?

“Automating the stages of app development”



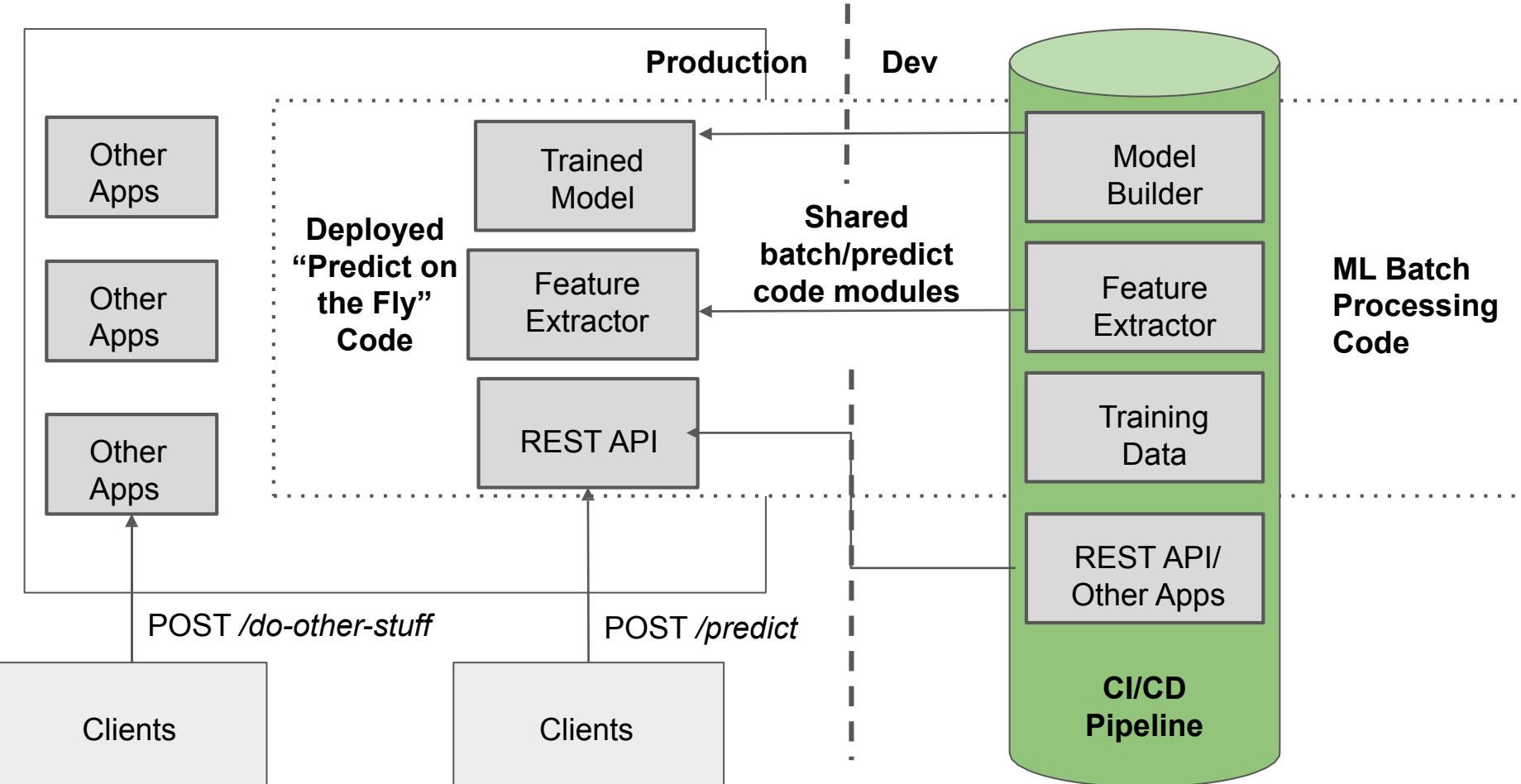
# Why Does This Matter?

Testing and deploying our applications according to the CI/CD method means:

- The system in an “always releasable” state
- Faster, regular release cycles
- Building and testing is automated
- Delivery and deployments are automated (at least to some extent)
- Visibility across the company (and audit log)



# What we will be Covering in this Section



# Why CircleCI?

In this course we will be using CircleCI

- Hosted platform
- Easy github integration
- One free project
- Many great features
- Trusted by many top companies



# Alternatives CircleCI

There are many amazing CI/CD platforms

- Jenkins
- Travis CI
- Bamboo
- Gitlab CI
- Team City
- Etc.

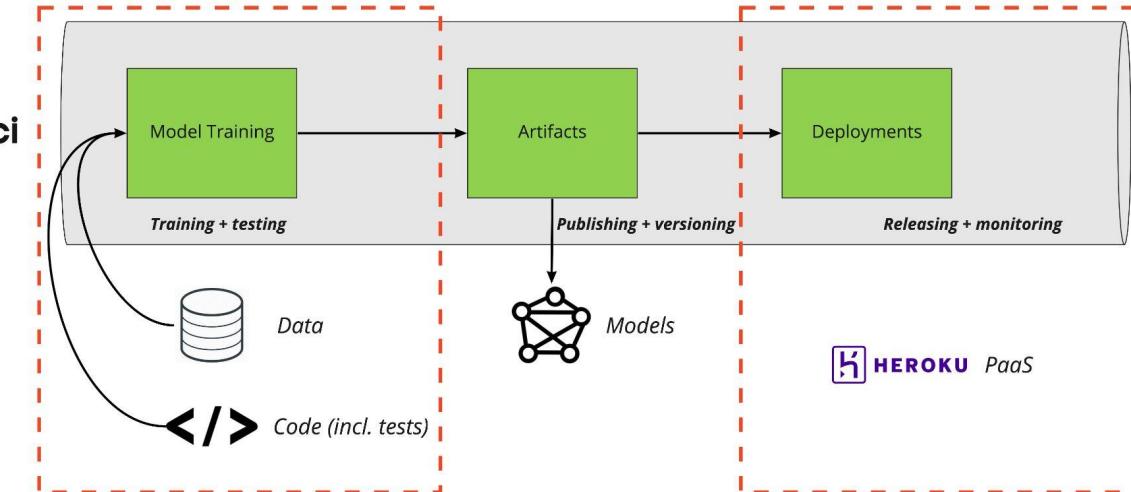
It doesn't really matter which one you use - so long as you use one!

# Let's Get Started!

See you in the next  
section



# Understanding CI/CD Automation 1



## Step 1: Automate the training, testing and deployment

*Before (section 5)*

```
tox -e train
```

```
tox
```

*Before (section 6)*

```
git push heroku main
```

# CircleCI Environment Variables Required

HEROKU\_API\_KEY

HEROKU\_APP\_NAME

PIP\_EXTRA\_INDEX\_URL (explained later in the section)

The screenshot shows the CircleCI web interface for project settings. The URL in the browser bar is [app.circleci.com/settings/project/github/ChristopherGTest/deploying-machine-learning-mod...](https://app.circleci.com/settings/project/github/ChristopherGTest/deploying-machine-learning-mod...). The page title is "Project Settings" for the repository "deploying-machine-learning-models". On the left, a sidebar lists "Overview", "Advanced", "Environment Variables" (which is selected and highlighted in grey), "SSH Keys", "API Permissions", and "Jira Integration". The main content area is titled "Environment Variables". It explains that environment variables allow adding sensitive data like API keys to jobs without committing them to the repository. It also mentions contexts for sharing variables across projects. At the bottom, there's a table with columns "Name" and "Value", and buttons for "Add Environment Variable" and "Import Variables".

Project Settings  
deploying-machine-learning-models

Overview

Advanced

Environment Variables

SSH Keys

API Permissions

Jira Integration

Environment Variables

Environment variables let you add sensitive data (e.g. API keys) to your jobs rather than placing them in the repository. The value of the variables cannot be read or edited in the app once they are set.

If you're looking to share environment variables across projects, try [Contexts](#).

Name	Value

Add Environment Variable Import Variables

# Heroku Environment Variables Required

PIP\_EXTRA\_INDEX\_URL (explained later in the section)

The screenshot shows the Heroku dashboard interface for an application named 'dmlm-example'. The top navigation bar includes links for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. The 'Settings' tab is currently selected. The main content area displays 'App Information' with the following details:

Setting	Value
App Name	dmlm-example
Region	Europe
Stack	heroku-20
Framework	Python
Slug size	165.8 MiB of 500 MiB
Heroku git URL	<a href="https://git.heroku.com/dmlm-example.git">https://git.heroku.com/dmlm-example.git</a>

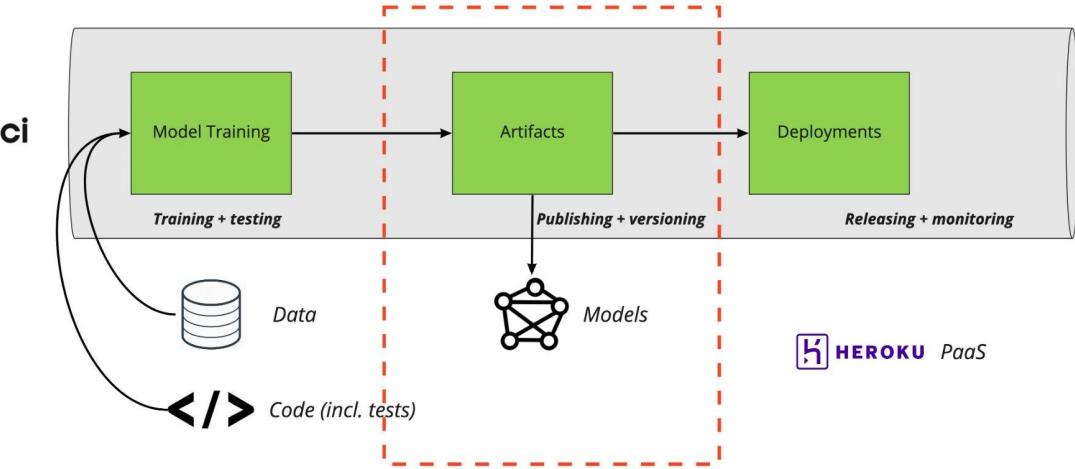
Below this, there is a 'Config Vars' section with a note: 'Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.' A 'Reveal Config Vars' button is present.





# Understanding CI/CD Automation 2

## Step 2: Automate the model building & publishing



*Before (section 5)*

```
tox -e train  
  
py -m build  
  
dist/tid-regression-model.tar.gz  
  
dist/tid-regression-model.whl
```

# Understanding Private Python Package Index Servers



# PIP\_EXTRA\_INDEX\_URL

master ▾

[deploying-machine-learning-models](#) / [section-07-ci-and-publishing](#) / [house-prices-api](#) / [requirements.txt](#)



ChristopherGS section 07 update (#742) [X](#)

Latest c

1 contributor

10 lines (9 sloc) | 246 Bytes

```
1 --extra-index-url=${PIP_EXTRA_INDEX_URL}  
2  
3 uvicorn>=0.11.3,<0.12.0  
4 fastapi>=0.64.0,<1.0.0  
5 python-multipart>=0.0.5,<0.1.0  
6 pydantic>=1.8.1,<1.9.0  
7 typing_extensions>=3.7.4,<3.8.0  
8 loguru>=0.5.3,<0.6.0  
9 # fetched from gemfury  
10 tid-regression-model==4.0.2
```

# CircleCI Environment Variables Required

PIP\_EXTRA\_INDEX\_URL

KAGGLE\_KEY

KAGGLE\_USERNAME

GEMFURY\_PUSH\_URL

The screenshot shows the CircleCI web interface for project settings. The URL in the browser bar is [app.circleci.com/settings/project/github/ChristopherGTest/deploying-machine-learning-mod...](https://app.circleci.com/settings/project/github/ChristopherGTest/deploying-machine-learning-mod...). The page title is "Project Settings" for the repository "deploying-machine-learning-models". On the left, a sidebar lists "Overview", "Advanced", "Environment Variables" (which is selected and highlighted in grey), "SSH Keys", "API Permissions", and "Jira Integration". The main content area is titled "Environment Variables". It explains that environment variables allow adding sensitive data like API keys to jobs without committing them to the repository. It also mentions contexts for sharing variables across projects. A table at the bottom shows a single row with "Name" and "Value" columns, both currently empty. Buttons for "Add Environment Variable" and "Import Variables" are visible.

app.circleci.com/settings/project/github/ChristopherGTest/deploying-machine-learning-mod...

Project Settings  
deploying-machine-learning-models

Organization Settings X

Overview

Advanced

Environment Variables

SSH Keys

API Permissions

Jira Integration

## Environment Variables

Environment variables let you add sensitive data (e.g. API keys) to your jobs rather than placing them in the repository. The value of the variables cannot be read or edited in the app once they are set.

If you're looking to share environment variables across projects, try [Contexts](#).

Name	Value

Add Environment Variable Import Variables

# Heroku Environment Variables Required

PIP\_EXTRA\_INDEX\_URL

The screenshot shows the Heroku dashboard interface for an application named 'dmlm-example'. The top navigation bar includes links for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. The 'Settings' tab is currently selected. The main content area displays 'App Information' with the following details:

Setting	Value
App Name	dmlm-example
Region	Europe
Stack	heroku-20
Framework	Python
Slug size	165.8 MiB of 500 MiB
Heroku git URL	<a href="https://git.heroku.com/dmlm-example.git">https://git.heroku.com/dmlm-example.git</a>

Below this section is a 'Config Vars' section containing the following text:

Config vars change the way your app behaves.  
In addition to creating your own, some add-ons come with their own.

A button labeled 'Reveal Config Vars' is located at the bottom of this section.



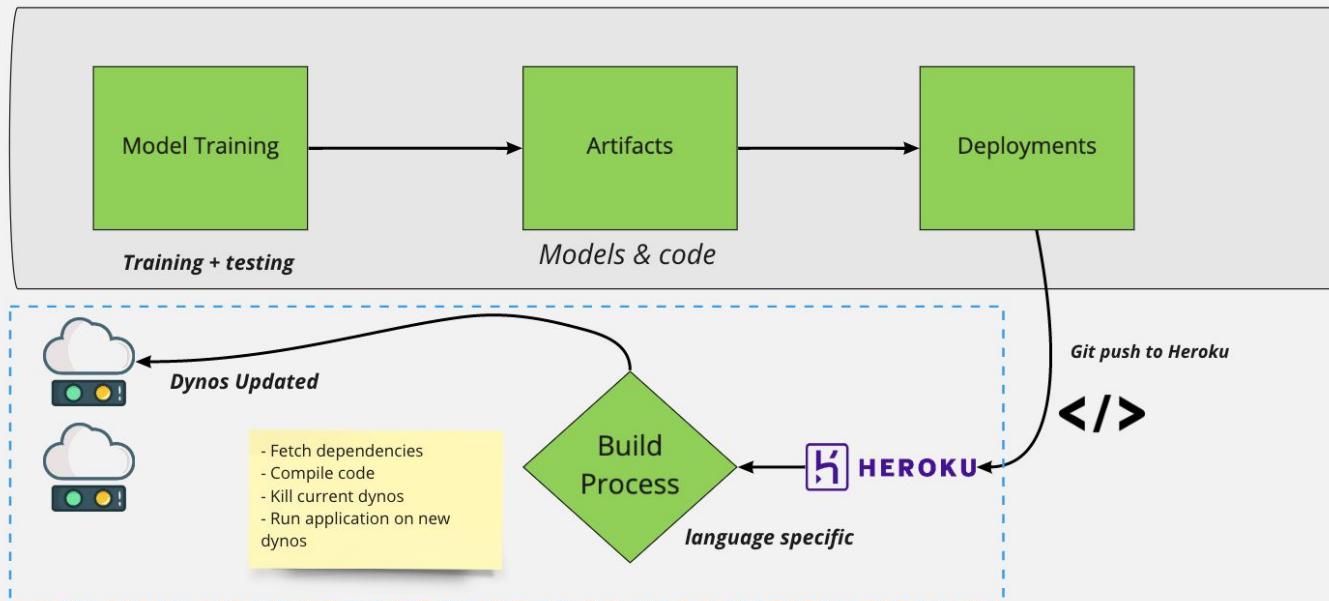


# Understanding Container Deployments

# What we saw in the Previous Section (7)

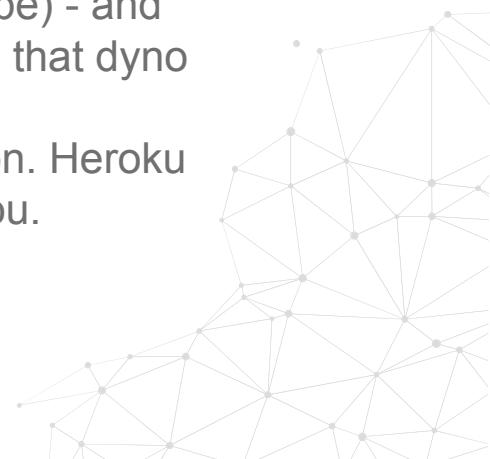


Code Deployment  
(Section 7)

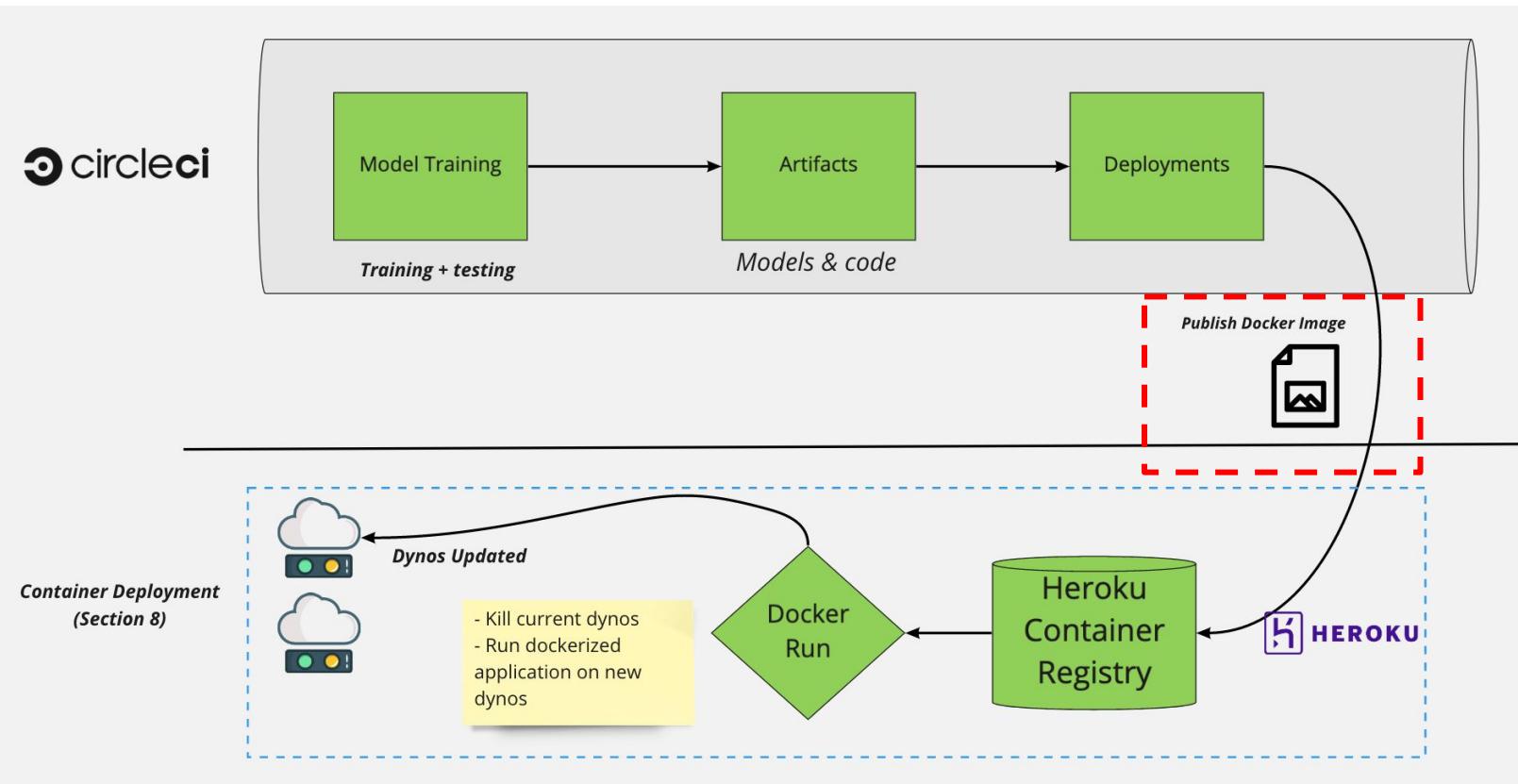


# What's a Heroku Dyno?

- All Heroku applications run in a collection of lightweight Linux containers called **dynos**.
- Web dynos are dynos of the “web” process type that is defined in your **Procfile**. Only web dynos receive HTTP traffic from the **routers**.
- Once a web dyno is started, the **dyno formation** of your app will change (the number of running dynos of each process type) - and subject to dyno lifecycle, Heroku will continue to maintain that dyno formation until you change it
- You can configure multiple dynos, which run in a formation. Heroku takes care of much of the maintenance and scaling for you.



# What We'll See in Section 8



**As in previous section, these commands will be run in CI - so pay attention to the .circleci/config.yml file**



# Section 9: Differential Tests

## Introduction





# What is a Differential Test?

A type of test that compares the differences in execution from one system version to the next when the inputs are the same. 

- Sometimes called “back-to-back” testing
- Very useful for detecting machine learning system errors that do not raise exceptions.
- Tuning them is a balancing act (depends on business requirements).
- Can prevent very painful mistakes that are not detected for long periods of time.

# Let's Get Started!

See you in the next section

# Section 9

Wrap Up





# Key Learning Points

- Differential tests can protect you from costly mistakes.
- Run them like any other tests in your CI pipeline
- Easiest with Docker

py**test**

# Other Tests to Consider



py**test**

- If a prediction and/or training speed is a key metric for you, consider implementing benchmark tests to compare the speed of functions from one system version to the next.
- A great framework for doing this is pytest-benchmark: <https://github.com/ionelmc/pytest-benchmark>
- In complex microservice environments, you may also wish to consider API contract testing.
- A nice framework for this is Pact:  
<https://docs.pact.io/>

# Section 10: Deploying to a Platform as a Service (Heroku)

## Introduction



# What is a Platform as a Service (PaaS)?

“A cloud-based service that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.”

# Understanding the Range of Possibilities

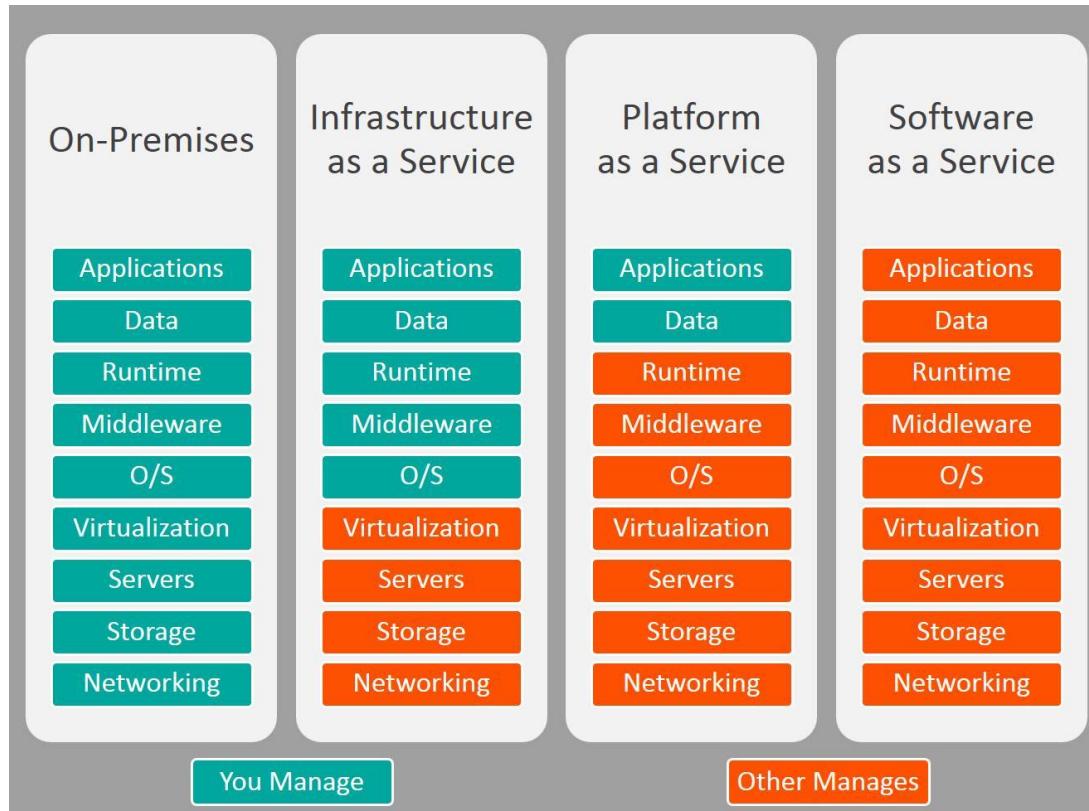


Image copyright:  
<https://www.bmc.com>

# PaaS Pros and Cons

Pros	Cons
Simple to setup, maintain and deploy	Hard / impossible to scale to a very large size
Easy to scale to moderate size	Tends to be more expensive than IaaS
Allows developers to focus on apps	Vulnerable to PaaS downtime
Easy creation of dev/test environments	Limitations on configuration

# PaaS Providers

- AWS Elastic Beanstalk
- Windows Azure
- Heroku
- Force.com
- Google App Engine
- Apache Stratos
- OpenShift
- PythonAnywhere
- ...and many more.



# Why Heroku in this Course?

- Note that in the IaaS section, we will use AWS ECS
- Heroku is very easy to use
- We can use one Heroku Dyno for free (ideal for teaching).
- Nice 3rd Party Add on options
- Works with Docker
- Great documentation
- Supports multiple languages

# Let's Get Started!

See you in the next section

# Section 10

Wrap Up





# Key Learning Points

- Integrate your deployments into your CI pipeline
- Deployments can be easy!
- Get used to checking deployments and logs - debugging in production is an important skill.

# Heroku Scaling



- On a paid plan, you can manually change the number of Heroku “dynos” to scale your app.
- On Performance Tier Dynos, you can enable and configure autoscaling:  
<https://devcenter.heroku.com/articles/scaling#autoscaling>



# Heroku Next Steps

- You can add databases, message queues and caches very easily, checkout:  
<https://data.heroku.com/> (but this will incur cost).
- It's worth exploring the heroku add ons:  
<https://elements.heroku.com/addons>
- If you are intending to use a monorepo with Heroku, you will need to follow this guide:  
<https://elements.heroku.com/buildpacks/lstoll/heroku-buildpack-monorepo>

# See you in the next section!



# Section 11: Deploying with Containers (Docker)

## Introduction



# What We Will Be Covering



- What are containers? What is Docker? (this section)
- Why use Containers and Docker? (this section)
- Installing Docker
- Configuring Docker
- Basic Docker Demo Locally
- Deploying Docker Images to Heroku

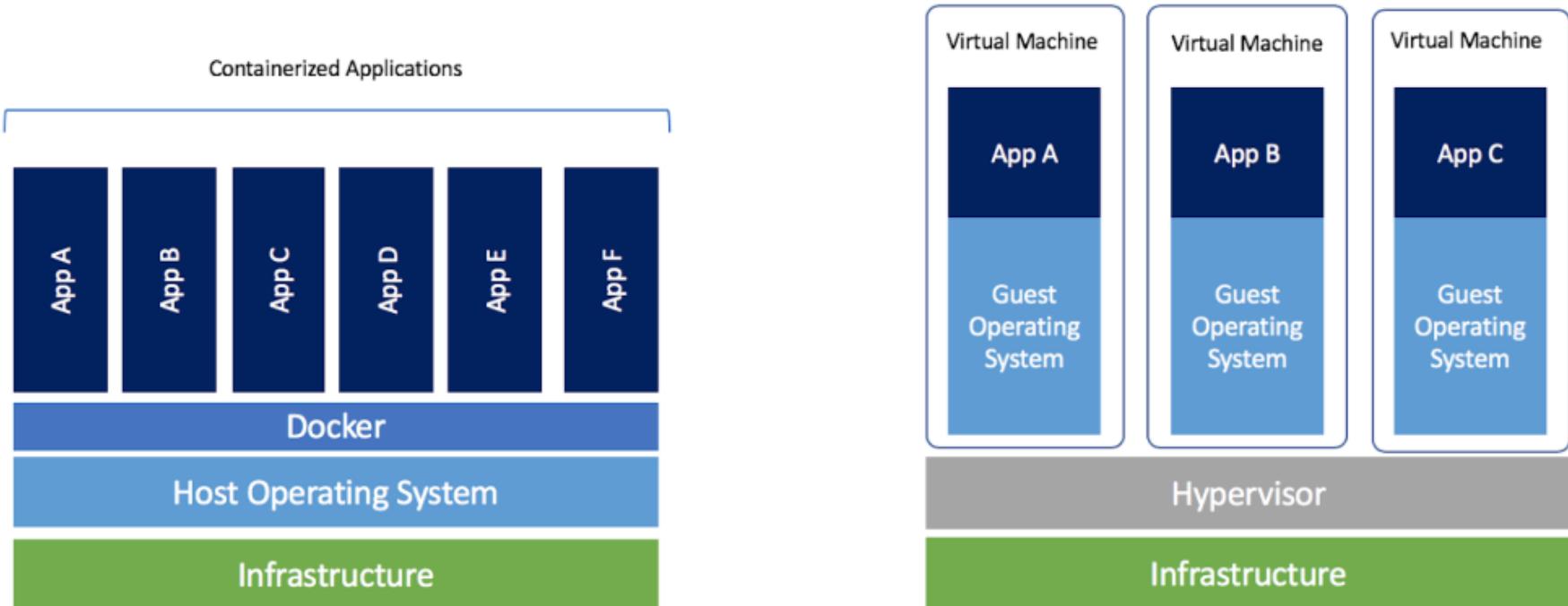
# What is a Container?

“A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.”

# What is Docker?

- Put simply, Docker is a tool to make creating, deploying and running containers easy.
- Docker is open source
- Released in 2013
- A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc.
- Containers are created from a read-only template called an image

# Containers vs. Virtual Machines



# Why Use Containers?



- Reproducibility
- Isolation
- Simplicity of environment management (Great for making staging/UAT match production)
- Ease of continuous integration
- Much faster and more lightweight than a VM
- Container orchestration options (e.g. Kubernetes)
- Docker is the most popular tool for creating and running containers

# Let's Get Started!

See you in the next section

# Section 11

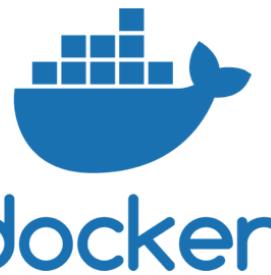
Wrap Up





# Key Learning Points

- Containerizing your applications brings a number of benefits: Reproducibility, isolation, ease of environment management and more.
- Not a silver bullet - not suitable for every use case (e.g. complex multiple OS requirements).
- Integrate your image building and deployments into your CI pipeline



# Docker Next Steps

- Have a browse of some of the open source Images on dockerhub: <https://hub.docker.com/>

# See you in the next section!



# Section 12: Deploying with IaaS (AWS ECS)

## Introduction



# What We Will Be Covering



- Overview of IaaS/AWS (this lecture)
- Risks and costs of using AWS
- Overview of ECS
- Creating your AWS account and permissions
- The Elastic container registry (ECR)
- Configuring your ECS Cluster
- Deploying to your cluster
- Automation

# What is Infrastructure as a Service (IaaS)?

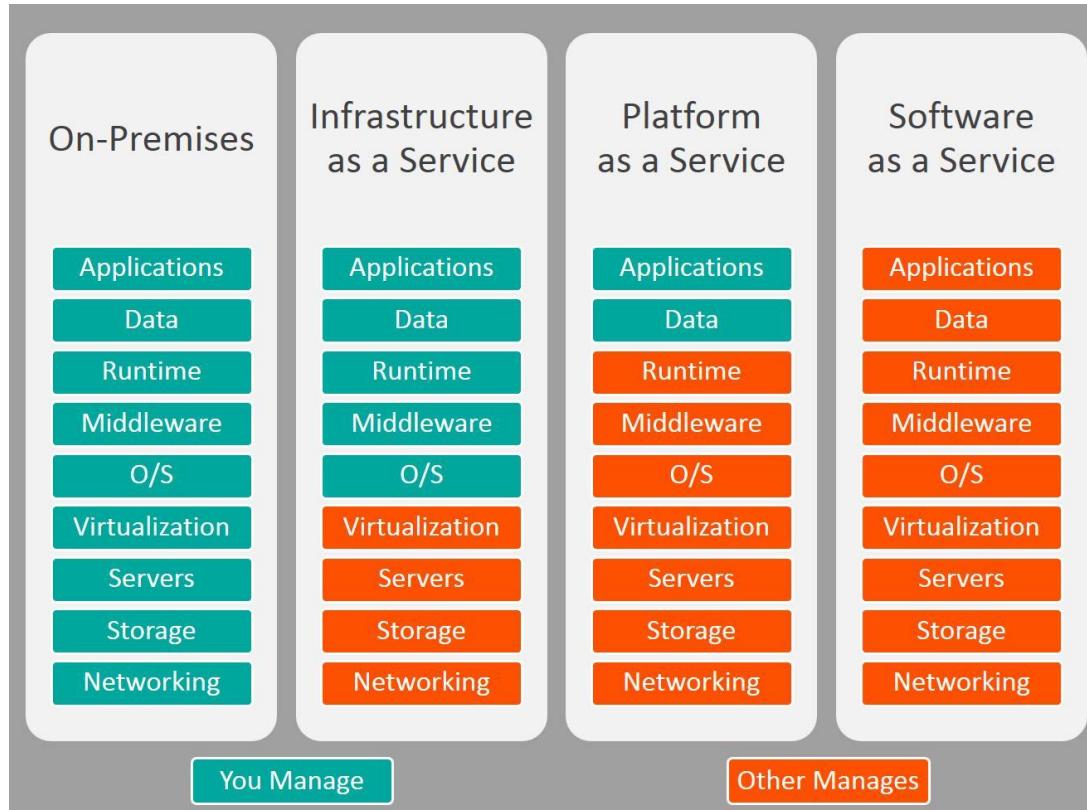


Image copyright:  
<https://www.bmc.com>

# What is Amazon Web Services?

- Amazon's cloud computing services
- Largest provider globally
- Vast range of services

# Section 12.2: AWS Pricing and Warnings



# Costs for Using AWS

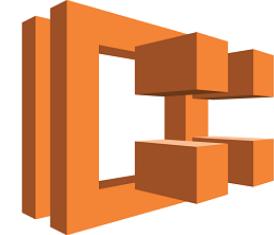
- This section of the course will incur some modest costs (less than USD 5) \*if\* you adhere to the following rules:
  - Delete your clusters after use (there is a lecture demonstrating this).
  - Do not create large EC2 instances (stick to t2.micro)
  - Do not upload many images to the registry (although this will be pretty negligible)
- Be careful with security!

# Section 12.3: AWS Elastic Container Service (ECS)

## Overview



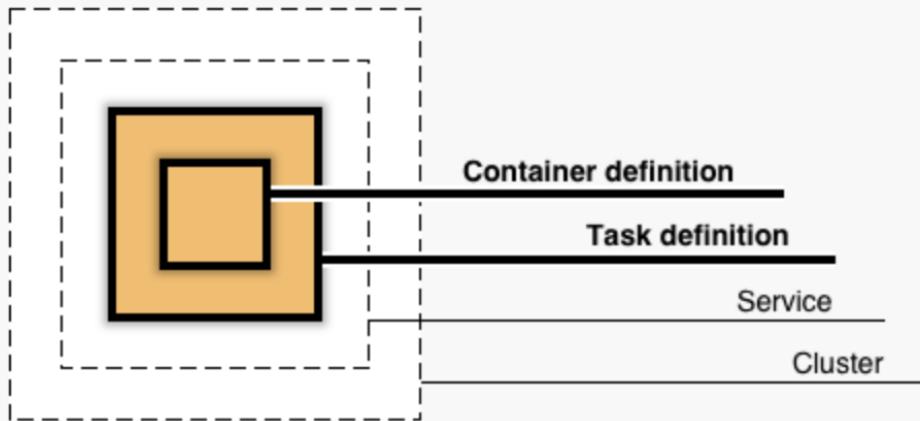
# What is ECS?



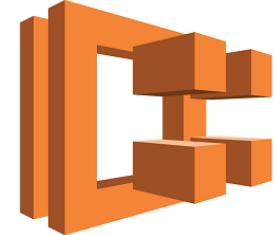
“Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster.”

# ECS Components

Diagram of ECS objects and how they relate

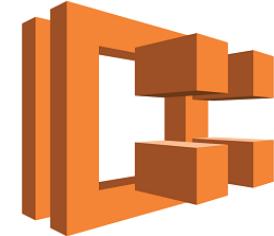


# ECS Task Definitions and Tasks



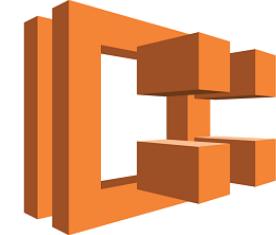
- A task definition is like a blueprint for your application (a bit like a Dockerfile). In this step, you will specify a task definition so Amazon ECS knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.
- Your entire application stack does not need to exist on a single task definition, and in most cases it should not.

# ECS Services



- Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. This is called a service.
- If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service depending on the scheduling strategy used.

# ECS Cluster



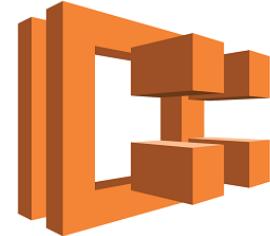
- An Amazon ECS cluster is a logical grouping of tasks or services.
- Two main launch types: Fargate and EC2:  
[https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch\\_types.html](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch_types.html)

# Let's Get Started!

See you in the next section

# Section 12.3b: Why ECS?





# What are your Options?

- Containerization has transformed how we deploy software
- Key options: Kubernetes, ECS, Docker Swarm
- Kubernetes is the container orchestration engine of choice...
- ...But has the most complex setup and management
- Self-Managed vs. Fully managed service
- Kubernetes is a good choice in many scenarios (but not all)
- New Player: Amazon Elastic Container Service for Kubernetes (EKS)

# Let's Get Started!

See you in the next section

# Section 12

Wrap Up



# Key Learning Points



- A range of container orchestration engine and managed/self-managed options
- Deploying to IaaS has far greater configuration complexity
- IaaS can scale to virtually any needs

# IaaS Next Steps

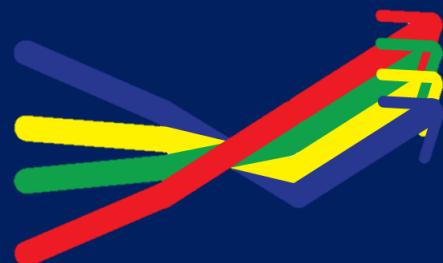


- Explore other AWS services: S3, Load Balancing, new ELK
- Explore other IaaS Providers: MS Azure, Google Compute Engine
- Explore other container orchestration engines (Kubernetes, Docker Swarm)

# See you in the next section!



# Deploying Models with Big Data



# What is big data?

Big data is essentially a large volume of data

- Structured
- Semi-structured
- Unstructured

Normally used for analytics and machine learning

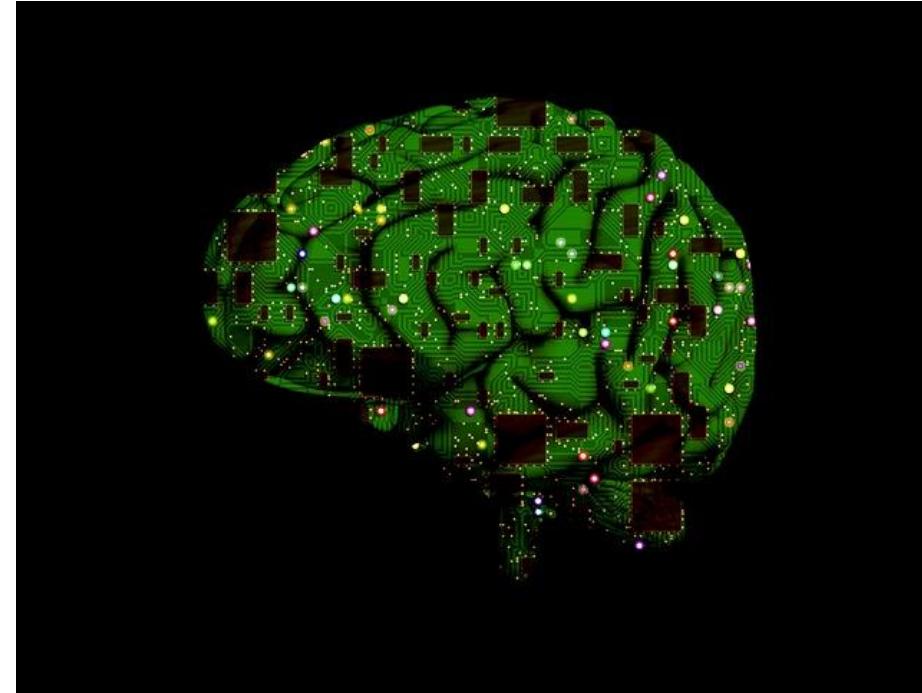
Typically terabytes and above



# Challenges of using big data in ML

## Big Data

- Text - large documents
- Images
- Complex Time Series



Deep Learning

# Challenges of using big data in ML

- Neural networks can be extremely compute heavy.
- During training takes up a lot of compute power.
- When scoring can also take a lot of computer power
- Difficult to scale
- Normally rely on GPUs.
- GPUs can be scarce and expensive

# Deployment pipeline for CNN

- Coding Challenges
- Deployment Challenges



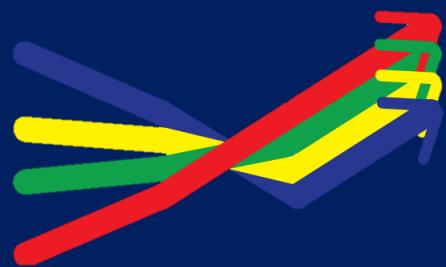
# V2 Plant Seedlings Dataset



# V2 Plant Seedlings Dataset

[Kaggle website](#)

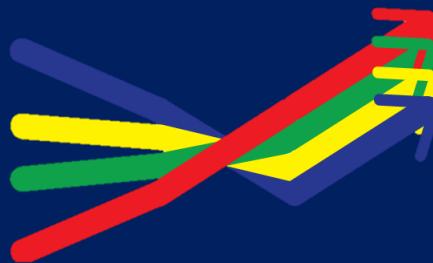
# CNN - Production Code



# V2 Plant Seedlings Dataset - Production Code

**Download and navigate the attached code to follow the video**

# Reproducibility in Neural Networks



# Reproducibility - why it matters

- Reproducibility ensures models' gain in performance are genuine.
  - Not from hidden sources of randomness
- Reproducibility reduces or eliminates variations when re-running experiments
  - Essential for testing and continuous integration and iterative refinement of models.
- Reproducibility is increasingly important as sophisticated models and real-time data streams push us towards distributed training across clusters of GPUs.
  - More sources of non-determinism behaviour during model training.

# What causes non-reproducibility?

- **Random initialization of layer weights:** the weights of the different layers are initialised randomly
- **Shuffling of datasets:** The dataset is randomly shuffled for example if we leave 10% of the samples for cross-validation within model.fit
- **Noisy hidden layers:** Dropout layers, which exclude the contribution of a particular neuron, are initialised randomly.
- **Changes in ML frameworks:** Different versions of ML libraries can lead different behavior.

# What causes non-reproducibility?

- **Non-deterministic GPU floating point calculations:** If using GPUs, certain functions in cuDNN, the Nvidia Deep Neural Network library for GPUs, are stochastic, which means that they are initialised randomly at each run.
- **CPU multi-threading:** CPU parallelization when using Tensorflow

# How to control for random initialisation?

- Keras gets its source of randomness from the NumPy random number generator
  - Seed the numpy random generator both for Theano or TensorFlow backend.
- Tensorflow uses multiple threads, which may cause non-reproducible results
  - Force TensorFlow to use single thread.
- **In python 3, you need to set a flag before running your script**
  - **PYTHONHASHSEED=0**
- **Set the cuDNN as deterministic**

# How do I do that?

- In the next article, I have included code from various sources that allows you to seed a neural network in its multiple initialisation instances