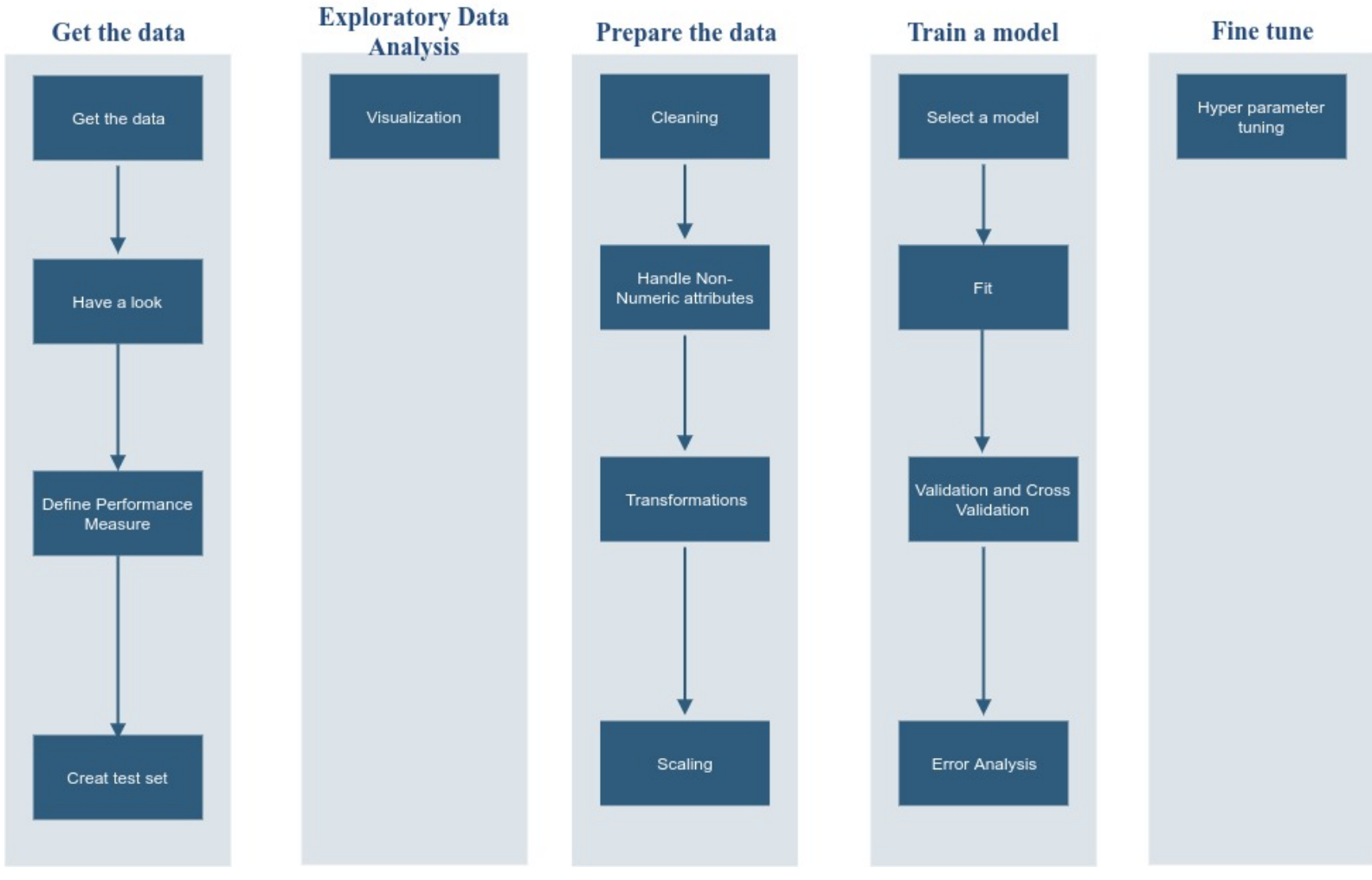
What is the ”Recipe” for Machine Learning

We will define a methodical approach to solve problems using Machine Learning.

This is the *Recipe for Machine Learning*


Get the data
Get the data
Have a look
Define Performance Measure
Creat test set
Exploratory Data Analysis
Visualization
Prepare the data
Cleaning
Handle Non-Numeric attributes
Transformations
Scaling
Train a model
Select a model
Fit
Validation and Cross Validation
Error Analysis
Fine tune
Hyper parameter tuning

There are no short-cuts !

Each step in the Recipe both prepares you for the next and, crucially, gives you *deeper insight* which improves the result.

Get the data

The first step is obtaining data for training and evaluation.

This is often the most challenging part !

- Interesting data is scattered: requires collection
- Supervised Learning requires labelled data; where do the labels come from ?

In this course, we will usually provide you with data so you will be mostly insulated from this challenge.

- Learning how to obtain data is a good skill to learn
    - Web scraping

Let's visit the notebook section Get the Data (Recipe_for_ML.ipynb#Recipe-step-A:-Get-the-data)

Look at the data

Always put your eyes on the data !

- You will learn about its "shape":
    - tabular ?
    - What are the attribute names ?
    - What are the types of the attributes ? Numeric ? Text ?
- You will learn about potential data problems
    - missing data
    - strange values

Don't even try to do anything with your data until you have at least the most basic understanding by performing an inspection.

Let's visit the notebook and [Look at the data](Recipe_for*ML.ipynb#Recipe-A.2:-Have-a-look-at-the-data*

Define a Performance Measure

How do I know how well my predictions are ? How do I know when to stop modelling and training ?

We answer these questions by defining a *Performance Measure* that quantifies how well your model is performing.

Let's visit the notebook and Define a Performance Measure (Recipe_for_ML.ipynb#select_performance_measure)

Performance Measure versus Loss Function

There may some confusion between the Performance Measure and Loss functions

- they are both evaluated over a set of examples
- they both measure performance of some sort

- A Performance Measure
    - is a property of the *problem* (not the model used to solve the problem)
    - you may have more than one Performance Measure
        - each expressing some desired quality of the prediction
    - is evaluated *out of sample*, that is, on non-training examples

- The Loss Function
    - is a property of a *model*: it guides a particular model's search for the best Θ
        - different models may have different Loss Functions
            - but the *problem's* Performance Metric is the same
    - is evaluated *in sample*, that is, on training data

Exploratory Data Analysis

This is one of the key steps of a good Data Scientist.

Besides "seeing" the data, we need to hear it: what is it telling us that may aid prediction

- any problems with the data that would inhibit learning ?
- any apparant relationship between target and a single feature ?
- any apparant relationship between target and combinations of features ?
- any apparant relationship between features ?
- what are the relative magnitudes of features ?

Often, understanding the data intimately can lead to

- transformations of the features that will aid prediction
- improved models

Let's visit the notebook and perform Exploratory Data Analysis (Recipe_for_ML.ipynb#Recipe-Step-B:-Exploratory-Data-Analysis-(EDA))

Prepare the data

It is not always the case that the data in "raw" form is adequate for modelling

- Cleanliness
    - dealing with missing data or anamolous values
- Numericalization
    - Converting non-numeric/categorical data into appropriate numbers
- Scaling, normalization
    - putting features on compatible scales
- Creating new "synthetic" features from original features
    - Knowing when/how to do this is what separates a good Data Scientist from an average one

We will call the process of preparing the data *Transformations* or *Feature Engineering*.

Transformation takes an example in raw form and creates a "processed" example suitable for modelling.

It is important to emphasize that "example" means either training, validation, or test.

Always apply transformations consistently to all example

- In particular: transformations applied to a training example should be applied to test examples at inference time

In [ ]:

Train a model

The model is our "predictor": the machine that takes features and produces predictions.

All the prior steps of the recipe were "prep-work": preparing the ingredients (data) for cooking (modelling)

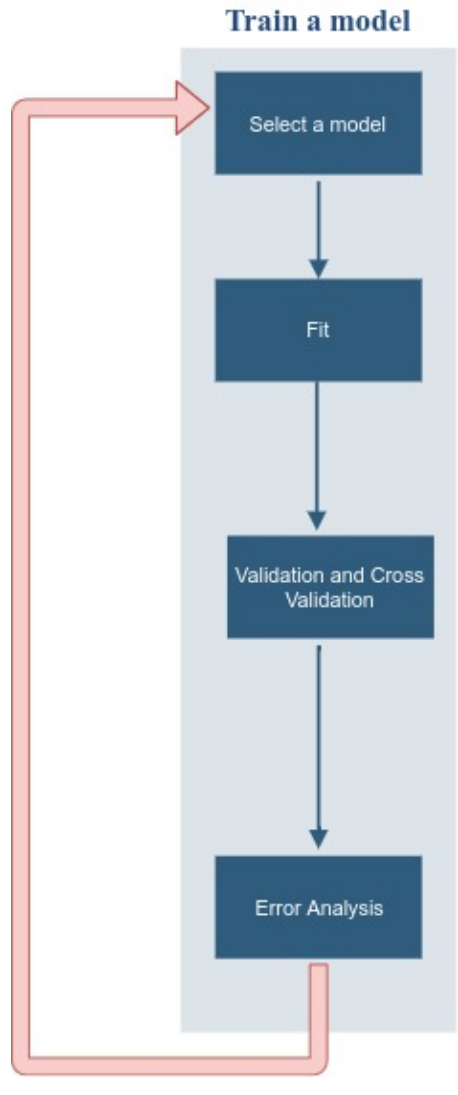
Unlike actual cooking, this step is *iterative*

- we try one model
- fit the model to the data
- examine the results critically
    - has the Performance Measure improved ? Is it good enough ?
    - learn lessons from errors
- improve the model and repeat

The iterative nature is often overlooked in the rush to learn models.

But Error Analysis is key to guiding us on the weaknesses of the existing model, and to improving the model.

Iterative training



Let's move to the notebook and Train the model (Recipe_for_ML.ipynb#Recipe-Step-D:-Train-a-model)

Error analysis

We have now fit a model and evaluated it with a Performance Metric.

If the metric is "good", are we done ?

Unfortunately, many people (and courses!) answer: "Yes"

This is discouraged; let's move to the notebook to Examine the errors (Recipe_for_ML.ipynb#Recipe-D.4:--Error-analysis) to see why a deeper analysis may be warranted.

Iterate: Linear Regression with higher order features

The Error Analysis we performed on our first model (single non-constant feature) suggested a need for improvement.

Two types of improvement come to mind

- Hypothesis iteration: try a different model
- Feature iteration: change/add to the features of the current model
    - adding a previously discarded feature
    - creating a synthetic feature

We will take the approach of adding a feature.

Let's extend Linear Regression with higher order features (Linear_Regression_HigherOrderFeatures.ipynb)

When to stop iterating

Adding second order features resulted in a perfect in-sample fit, so there is no point iterating further.

In general, this will not be the case.

How do we know that our model is "good enough" ?

Let's explore Bias and Variance (Bias_and_Variance.ipynb)

Fine tune

There are often "tweaks" that can be applied to a near-final model in order to squeeze out increase performance.

For example: many models have *hyper parameters*.

These are values that are *chosen* at model construction, rather than *discovered* by fitting during training (Θ)

- the k in K Nearest Neigbhors
- the degree d of the polynomial when constructing higher order features $\mathbf{x}^d$
- strength of the regularization penalty
- whether to include/exclude the interecept Θ_0 in a Linear Regression

Perhaps a different choice of a hyper-parameter would improve the model ?

We can try many choices before settling on the one giving the best Performance Metric.

Hyper parameters search is another reason for using Cross Validation

- we can't use the Test set more than once
- with a single Validation set: we might overfit to the validation set
    - that is, choose a value for the hyper parameter that is best for this *single* validation set

In [ ]:
```
print("Done !")
```