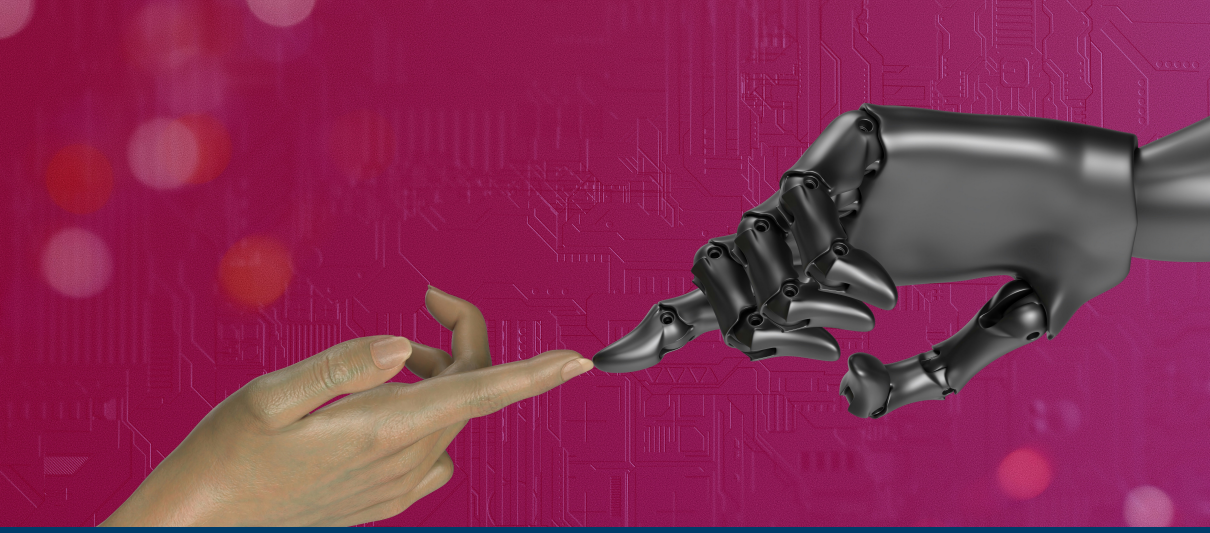


# Hands-On Machine Learning

Dr. Derek Bridge | 2026/2027

UCC Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

# 6. Preprocessing

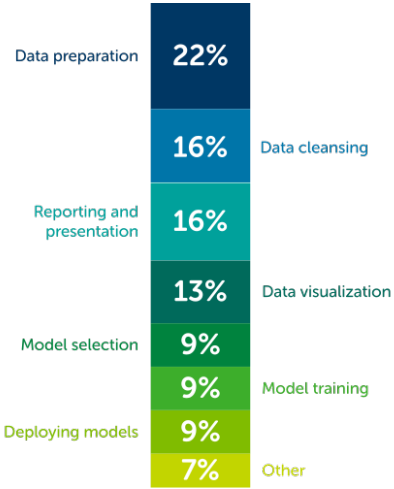
In this lecture, we predict whether to approve a loan or not. We use a slightly modified version of a dataset from Kaggle.

**Machine Learning Projects (simplified!)**

1. Understand the business problem.
2. Select performance measures.
3. Acquire a dataset.
4. Take a cheeky look.
5. Cleanup anything that is simply invalid.
6. Split into training and test sets using holdout.
7. Exploratory Data Analysis.
8. Feature Engineering.
9. Preprocess the data.
10. Model Selection.
11. Repeat steps 7–10 until satisfied.
12. Error Estimation: test on the test set.
13. Decide whether to deploy.
14. If yes, then train on the whole dataset.
15. Once deployed, monitor and retrain regularly.

How do data scientists spend their time?

(Survey of $m = 1,966$ respondents.)




**Step 1: Understand the business problem.**

We are a small company, making loans in a competitive market. We want to speed-up loan decisions and reduce the number of people who default on their loans.

- Notice that the business problem is not expressed in terms of Machine Learning, and ML will likely be just one component of the solution (assuming it gets used at all).
- Assemble a team: Computer Scientist(s), Statistician(s) but also Domain Expert(s).

**Step 2: Select performance measures.**

- This is Supervised Learning of a Binary Classifier.
- We will measure accuracy.
- We will compare with a **majority-class classifier** (a **dummy classifier**).

We might want a 'reference point' to know whether we have succeeded.

- E.g. maybe we want accuracy to reach some threshold, or to be some amount higher than human performance at this task or an existing computer system or a dummy classifier.
- E.g. maybe we must not only be better than current performance but **statistically significantly** better.

Sometimes there is more than one performance measure.

- E.g. maybe in regression we also want to measure overestimates separately from underestimates; in binary classification maybe we also want to measure false positives and false negatives.
- E.g. maybe we want to measure time/memory for training.
- E.g. maybe we want to measure time/memory for inference.
- E.g. maybe we want an incremental learning algorithm that can easily accommodate new training examples as they arise.
- E.g. maybe we care about learning an interpretable model or being able to explain the system's predictions (see later lecture).

**Step 3: Acquire a dataset.**

Dataset acquisition is more easily said than done — especially since we need a *labeled dataset*. In the real world, datasets don't come from your lecturer or from Kaggle. More on this in a later lecture.

**Step 4: Take a cheeky look.**

Enough to get an understanding of the kind of data but not too much — avoid leakage.

**Step 5: Cleanup anything that is simply invalid.**

Historical datasets may contain values that your domain expert says are invalid. Occasionally, your domain expert may be able to correct the invalid values. More usually, we delete rows and perhaps columns to get rid of these.

- E.g. if a feature value is required but there are examples where it is missing.
- E.g. if there is a maximum value for a feature but examples where this maximum is exceeded.
- E.g. if there are examples where a nominal-valued (non-numeric) feature has a value that falls outside of the finite set of allowed values.
- E.g. if there are examples where the target value is missing.
- E.g. if there are duplicate examples – these may or may not be invalid.

Only do this to data that you will henceforth disallow.

**Step 6: Split into training and test sets using holdout.**

Stratified holdout in this case!

**Step 7: Exploratory Data Analysis.**

- Best to perform EDA and try Feature Engineering on a copy of the training set.
- Visualizations: histograms, scatter plots, strip plots, box plots, ...
- Statistics including correlation coefficients.
- Some people drop features at this point, e.g. where there is multicolinearity or where features are not predictive of the target. But be wary of this.

**Step 8: Feature Engineering.**

In **feature engineering**, we invent new features that are computed from the existing features — hopefully, ones that are predictive of the target.

- E.g. products or ratios of existing features.
- E.g. the result of applying functions to existing features, such as squaring, taking the square root, taking the log, ...

We would then do some more EDA to see whether the new features are promising or not.

**Step 9: Preprocess the data.**

Among other things, we must consider:

**Outliers**

See next slides.

**Missing values**

See next slides.

**Scaling numeric-features**

See previous lecture.

**Converting non-numeric features to numeric**

See next slides.

**Feature selection/dimensionality reduction**

See later lecture.

Create a pipeline of scikit-learn transformers that puts all your preprocessing code in one place.

Also insert any promising features from Feature Engineering.

## Outliers

- Earlier, we deleted examples that had extreme values which our domain expert regarded as invalid.
- But we may still have **outliers**. If used in training, they may skew what gets learned — not so much Decision Trees, but many other models.

### Decide which are outliers

- Speak with the domain expert.
- Some people use the first and third quartiles (Q1 and Q2). Anything below 1.5 x Q1 or above 1.5 x Q2 is an outlier. But (a) this looks at features individually, instead of together, and (b) it may result in too many outliers.
- There are ML methods —often using unsupervised learning— for identifying outliers/anomalies, e.g. Isolation Forests.

### Decide what to do about them

- We can do nothing — leave them.
- We can delete examples that have outlier values.
- We can clip the values to some maximum if they're too big or minimum if they're too small.
- We can apply some mathematical function (e.g. log and sqrt will reduce the effect of large values).

It is hard to say whether these solutions should be applied only to the training set or also applied to validation/test sets.

## Missing values

- Most Machine Learning algorithms cannot handle **missing values**.
- We may have removed some of them during data cleaning in discussion with our domain expert. But some may remain.

### Detecting missing values

- They may appear in a dataset as NaN or None. (E.g. when missing from a csv file, Pandas reads them in as NaN.)
- But they may appear as zero or as "?" or "N/A" or "unknown" or...

### What to do about them

- We can delete examples that have missing values, or we can delete features if they are plagued by missing values.
- Or we can **impute** a value:
  - We can replace missing values by the mean (for numeric-valued features), or the mode (for nominal-valued features), or by some constant.
  - Or we could learn a model that predicts the missing value from the other features in the example.
- The mean/mode/etc. must be computed from the training data only.

## Converting non-numeric features to numeric

### Boolean-valued

- Two values. E.g. Education has values “Graduate” and “Not graduate”.
- Binary encoding: e.g. “Graduate” becomes 0 and “Not Graduate” becomes 1.

### Unordered nominal values

- More than two values.
- **Ordinal encoding**: assign integers, e.g. “Semiurban” 0, “Urban” 1, “Rural”’ 2.
- But ML algorithms might assume that the integers themselves are meaningful, when they’re actually arbitrary. E.g. an algorithm might assume that “Semiurban” (0) is more similar to ”Urban” (1) than it is to “Rural” (3).
- Warning: in many datasets, unordered nominal values have already been encoded in this way.
- Often, **one-hot encoding** is preferable.

**One-Hot Encoding**

- If the original nominal-valued feature has, e.g., 3 values, then we replace the feature by 3 *binary-valued* features.
- In each example, exactly one of the new features is set to 1 and the rest are zero.

| Loan_Amount | Loan_Term | Property_Area |
|---|---|---|
| 71.0 | 360.0 | Rural |
| 40.0 | 180.0 | Rural |
| 253.0 | 360.0 | Urban |
| 187.0 | 360.0 | Urban |
| 133.0 | 360.0 | Semiurban |

becomes

| Loan_Amount | Loan_Term | Semiurban | Rural | Urban |
|---|---|---|---|---|
| 71.0 | 360.0 | 0 | 1 | 0 |
| 40.0 | 180.0 | 0 | 1 | 0 |
| 253.0 | 360.0 | 0 | 0 | 1 |
| 187.0 | 360.0 | 0 | 0 | 1 |
| 133.0 | 360.0 | 1 | 0 | 0 |

- One-hot encoding turns each non-numeric feature into multiple binary-valued features. Thus, you can end up with a lot of new features. Maybe you need to do some dimensionality reduction afterwards.

## Converting non-numeric features to numeric

### Ordered nominal values

- More than two values but on some sort of scale, e.g. Dependents has values "0", "1", "2" and "3+".
- Ordinal encoding: "0" becomes 0, "1" becomes 1, "2" becomes 2 and "3+"' becomes 3.
- Why might it be wrong? What assumption does it make?
- Instead of guessing, we could compare One-Hot Encoding and Ordinal Encoding in our grid search.

**Step 10: Model Selection.**

- Choose between holdout and $k$-fold CV.
- Choose candidate models, Decision Trees, kNN,...
- For each model, create grids.
- Run.
- Analyse validation errors; check for underfitting/overfitting.

**Step 11: Repeat steps 7–10 until satisfied.**

Be guided by what you have learned, e.g. whether you are underfitting/overfitting.

**Step 12: Error Estimation — Test on the test set.**

**Step 13: Decide whether to deploy.**

**Step 14: If yes, then train on the whole dataset.**

**Step 15: Once deployed, monitor and retrain regularly.**

# Image credits

Image from Anaconda 2022 State of Data Science Report, 2022

Photo by The New York Public Library on Unsplash